



Graduation Thesis

LocFlow - Localisation

Rehan Ahmed

Supervisor - **Joey Teunissen** (Technical Director)

ProfitFlow.

Maagdenburgstraat 14

7421 ZC Deventer

+31640587446

Version 4.2

Acknowledgements

This report was written by Rehan Ahmed and this project was completed as a graduation project for the HBO-IT Software Engineering programme at Saxion University.

Throughout this graduation project, I have received a great lot of direction and support from the Technical director and my supervisor Joey Teunissen. I'd want to express my gratitude to ProfitFlow for offering such an opportunity and a challenging project.

The ProfitFlow development team is also to be commended. Joey Teunissen, Olivier Smit and Remy Tapper, in particular, provided invaluable input, assistance, and suggestions throughout the process.

Finally I'd want to express my gratitude to Jeroen Linssen, acting as my graduation teacher. His constant encouragement, feedback, and understanding, as well as his knowledge, enabled me to keep a high motivation and produce a valuable project with a good graduation report.

Table of contents

Acknowledgements	2
Table of contents	3
1. Introduction	5
1.1 Problem Statement	
1.2 Objectives	
1.3 Approach	
1.4 Main Research Question	
1.5 Deliverables	
1.6 Research Plan	
2. Organisation	11
2.1 ProfitFlow	
2.2 Structure	
2.3 Daily Process	
2.4 Limitations	
2.5 Risks	
2.6 Mitigation Strategies	
2.7 Quality Assurance	
2.8 Definition of done	
3. Research	15
Main Research Question	
3.1 What are the available existing solutions and how do they compare to each other?	
3.2 How can we develop our own platform?	
3.2.1 Problem Analysis:	
3.2.2 Survey	
3.2.3 Interview	
3.2.4 Requirements	
3.3 How can our solution be integrated into existing and new projects?	
3.4 How is it possible to integrate Google Translate into our own platform?	
4. Design	29
4.1 Architecture Design	

4.2 Business Design	
4.3 Database design	
4.4 Frontend Mockup	
4.5 Styling	
5. Realisation	34
5.1 Database:	
5.2 Backend:	
5.3 Hasura:	
5.4 Frontend:	
5.5 Services:	
5.7 Steps during realisation:	
6. Testing	42
6.1 Database and Backend:	
6.2 Front-End UI testing:	
6.3 Translate Service testing:	
6.4 NPM Integration testing:	
7. Advise	45
7.1 Development Process	
7.2 Testing Process	
7.3 Scaling	
7.4 Continuation	
8. Conclusion	48
References	49
Glossary	50
Versioning	51

1. Introduction

This is a Thesis for the Graduation Project. This graduation is for Bachelors of Software Engineering with specialisation in Big Data. The internship for the Graduation project is performed at ProfitFlow. B.V Deventer.

ProfitFlow is a rapidly expanding company that specialises in process digitalisation and optimisation for MKB companies. This mostly includes back-office automation, which can include planning and asset management. These applications are always web-based and often require integration with ERP packages such as Exact and Unit4. Apart from this, ProfitFlow creates software as a service platforms in collaboration with other companies that hold the domain expertise. These platforms often come from previous customer software builds and are mostly focused on expanding the created software to other companies in the same field.

As the company is still in its growth phase, there are not many departments. There are three departments: management, human resources, and software development. The software development department was hosting this graduation project. In terms of employment numbers, the software development department is the largest. The majority of the company's work is done on-site, however employees have the option to work remotely.

The majority of the websites built for customers use NextJS as the front-end, Hasura as the back-end, and PostgreSQL as the database. However, depending on the needs of the consumer, alternative front-end, back-end, and database technologies can be employed.

Many customers want websites and apps to be available in a variety of languages. Internationalisation, also known as i18n, is the process of making a technology product available in multiple languages. At the start of the graduation ProfitFlow's i18n was done in a less efficient manner. The i18n files for each project were saved within a project's Gitlab repository.

This is where the software development project for this Graduation came in. The goal was to investigate the existing methods for i18n development and utilisation, as well as to improve the i18n development process. Furthermore creating a platform that serves as a translation directory. This platform now is useful both during development and while using i18n in projects.

1.1 Problem Statement

ProfitFlow develops technological solutions for its clients. Websites, mobile apps, B2B portals, dashboards, and more products are available. Many clients desire that their product be available in multiple languages. Internationalisation, or i18n, is the process of making multiple languages available in an application.

The NPM i18n library is mostly used to implement internationalisation. The keys and translations are stored in a project, inside a JSON file. These files enable projects to support several languages. Each supported language has its own directory with JSON files of keys and translations. [1][2]

The problem, however, appeared when managing these translations. For instance, if you need to update a translation or add support for a new language. Because there are multiple projects, you had to go into the unique repository of that project and make changes to that specific file. Furthermore, if you needed to add a new key, you would need to add the key with their translations to all the separate language directories manually.

The idea was to develop a platform that can handle all of the projects and their translations in one location. But then there was the problem of figuring out how to incorporate the platform's translations into existing and new projects.

1.2 Objectives

As it is apparent from the problem statement, the main objective of this project was to centralise the management of translations. The objective was to research existing solutions, taking into account their cost, scalability, and usability aspects. Based on the research, it was reasonable to draw up an advisory report on things to consider while developing our own platform.

Following the initial research and the advisory report the objective was to develop a platform as a solution. A full software development cycle was carried out during the development stage. With all of the software development processes, such as stakeholder requirement analysis, technology research, planning, development, integration, and testing,

Another objective which was decided at the start after the stakeholder analysis, was to find a way to include Google Translate into the Internationalisation process, to quickly and easily generate translations for the keys.

Before the start of this graduation, the translations were done manually by the software engineers for each project inside the relative project repository. There had been no work done to centralise internationalisation of multiple projects. So the goal to develop a software solution, was a goal to drive the organisation to be more efficient and organised in its translation management for all of the projects that it oversees.

1.3 Approach

This was a software development project. Therefore at the initial stage there were different research methods used to create a better understanding of the project. Subsequently there was software development of a Proof of Concept application, based on the requirement analysis from the research stage.

Research methods:

The research was conducted using different research methods. There are two main types of research methods Qualitative and Quantitative. In addition as this research was to aid development of a software product, the DOT Framework was used as a guide for research methods. [3]

Qualitative research is about getting the data from experiences, emotions or behaviours. The methods include Interviews, Focus Groups, on-site Observations.

Quantitative research is when you get numerical data which can be ranked or measured. The common methods for this type of research are, surveys, library research or experiments based on a hypothesis.

The Development Oriented Triangulation (DOT) framework helps in structuring and communicating for an ICT product based research. [4]

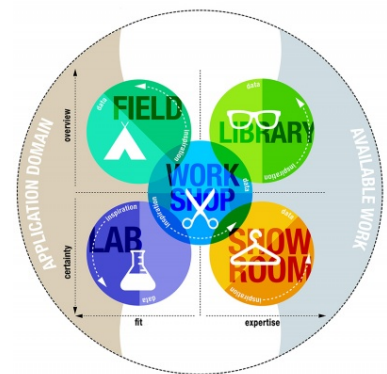


Fig 1: DOT Framework

First of all at the start of the graduation some research questions were drawn. Then to answer the research questions, a mix of the research methods from DOT framework were used.

After getting a transparent picture of the needs of the project from the research methods, some requirements in a MoSCoW format were formed in collaboration with the stakeholders.

Development Stage:

After research the next phase was the development stage. In the development stage taking the requirement analysis into consideration, a project backlog was prepared. For the development stage the main approach was using scrum as a framework.

SCRUM:

Scrum is a framework for project management with iterative cycles. One cycle is called a sprint, it usually lasts for 2 weeks. For every sprint the stories/tasks are planned before the start of the sprint. During the sprint work is done to complete all the planned tasks. Towards the end of the sprint a sprint-planning takes place, to decide the tasks for the next sprint. At the end of a sprint there is a sprint retrospective, this is a meeting where it is discussed what went well during the previous sprint and what can be improved in the upcoming sprint. [5]

In this instance the early weeks of the project were focused on research and design of the architecture, database and web-app. Following that, the scrum process started, where the sprint goals were focused towards programming the solution. And finally in the final phase of the project there was testing, integration and documentation.

The sprints for development started with planning for which stories from the backlog were to be worked on during this sprint. And ended with a demo of the progress to the development team. From the demo, feedback was collected about things to be changed, which was then translated into tasks as stories. Finally, those stories were planned for upcoming sprints.

Apart from that at the daily stand-ups the tasks that were currently in progress were mentioned. And if some help or meeting was required it was also notified during the standup.

1.4 Main Research Question

During the Plan of Approach the research questions were defined. At the time, the only information available was the initial description of the project. According to the description, the purpose was to centralise translation management. As a deduction, the main question was formed:

- **How can we centralise the management of translations for many projects?**

Following the definition of the main question, I brainstormed ways to best answer the question and develop a solution. The problem was broken into smaller segments during this time. Because the brainstorming created questions regarding various aspects of the project, these questions were defined as subquestions to the main question.

Subquestions:

- What are the available existing solutions and how do they compare to each other?
- How can we develop our own platform?
- How can our solution be integrated into existing and new projects?
- How is it possible to integrate automated translations into our own platform?

1.5 Deliverables

The deliverables produced by the conclusion of the project are following:

- Advise Report
A report consisting advise about the competitors solutions and how they compare to each other. Advise on development workflow and integration of the platform. Advice in the end about the continuation of the Project
- Requirement analysis
A report about the project requirements derived from the research and analysis of the requirements
- Front-End
Codebase of the main front-end application
- Back-End
Codebase of the backend connected to front-end and database
- Database
Codebase of the Database and schema designed for the solution of this problem
- NPM Library
Codebase of NPM Library used for integration of projects with the platform.
- Google Translate Service
Codebase of Google API integrated into the Front-end application.
- Technical Documentation
Documentation of the different codebases and the complicated parts of the code in more detail.

1.6 Research Plan

In the previous subsections, the research questions and deliverables were defined. Taking those research questions and deliverables into consideration, I created a research plan. To connect the research questions to deliverables, I planned some research methods which helped to produce the deliverables. In the following table, the link between them can be seen:

Research Question	Research Method	Deliverables
What are the available existing solutions and how do they compare to each other?	Library research <ul style="list-style-type: none">- Literature study- Available product analysis- Best good and bad practices	<ul style="list-style-type: none">• Advise Report
How can we develop our own platform?	Field research <ul style="list-style-type: none">- Problem Analysis- Interview- Survey- Explore user requirements Workshop <ul style="list-style-type: none">- Brainstorming- IT architecture sketching- Code implementation & Prototyping	<ul style="list-style-type: none">• Requirement analysis• Front-End• Back-End• Database• Technical Documentation
How can our solution be integrated into existing and new projects?	Field research <ul style="list-style-type: none">- Explore user requirements- Task Analysis Workshop <ul style="list-style-type: none">- IT architecture sketching- Code implementation & Prototyping Lab research <ul style="list-style-type: none">- Usability testing	<ul style="list-style-type: none">• Front-End• Technical Documentation• NPM Library
How is it possible to integrate Google Translate into our own platform?	Library research <ul style="list-style-type: none">- Literature study- Best good and bad practices Workshop <ul style="list-style-type: none">- Code implementation & Prototyping	<ul style="list-style-type: none">• Advise Report• Google Translate Integration• Technical Documentation

Table 1: connecting research questions, methods and deliverables

2. Organisation

2.1 ProfitFlow

ProfitFlow is a small scale startup with a 2 years history. It is a rapidly growing company, at the moment there are 3 departments namely the development, HR, and the management. These department are made up of 15 people.

ProfitFlow specialises in MKB process digitalisation and optimisation. They help in Back-office automation which are largely concerned with planning and asset management. These applications are always web-based and frequently require interaction with ERP systems like Exact and Unit4. The company also develops software as a service platforms in conjunction with other firms with subject expertise. [6]

2.2 Structure

Within ProfitFlow there is no hierarchical approach to the team. Instead the development team is subdivided into sub teams. These sub teams are provided with their own project and consists of at least two to three members:

1. Frontend engineer
2. (Senior) Frontend engineer
3. Backend engineer

The ideology of this method is that this way there is a self sustaining team where the team itself is capable of making all the choices of their own project. Internal discussions are made within the team and all choices are up to the team in collaboration with the CTO

Sometime the team is assigned a project manager, mostly the backend engineer due to the lesser workload compared to the frontend. As the backend engineer implements the data model, it is his responsibility to keep track of the functionality and overall progress of the project.

Besides this, the team, if possible, could also be responsible for direct communications with the customer to ensure that there are no communications barriers, that could be found by adding an overarching project manager who knows less about the current project than the team itself. In some cases an overarching project manager will do the communications if the team it self deems unfit to suit this role.

2.3 Daily Process

Work starts at 9:00 in the morning, the day starts with a daily-standup at 9:30. At the daily-standup everyone tells what they did yesterday, what they'll do today and finally if there is something that is blocking their work.

Following that, everyone gets to work on the current task that has been assigned to them. The scrum process is managed using Jira. As a result, Jira is used to track the tasks/stories assigned to each developer.

This graduation project had its own Jira board, which was managed throughout the project and updated during each sprint planning. After conducting research and planning, the stories were written to a backlog, but different stories were added or removed during the sprint depending on the project goals.

Because the supervisor is usually on-site and ready for assistance, he was reachable immediately whenever an inquiry, feedback, or assistance was required. Other senior developers at the company were also open for assistance as needed.

2.4 Limitations

The general limitations that you must account for during the project life cycle are known as project constraints. A financial limitation, for example, means you're limited to a set project budget, whereas a time constraint means you have to finish your project within a certain amount of time.

Cost:

To develop the project, the available resources at the company were to be used. The technologies the company already had Licenses for. If a technology is very important and needs to be paid for, it could be discussed with the supervisor.

The goal was to keep the cost of managing the translations below the cost for Licensing the existing alternative platforms mentioned in the competitor analysis.

Time:

Time management is very important for the success of a project. So the planning had to be done keeping in mind the overall project timeline. For this project the available time was the duration of the graduation period which was 21 weeks starting from 07-February-2022 to 01-July-2022.

2.5 Risks

Project risks are any unexpected mishaps during a project which can affect the budget, timeline, quality or completion of the project.[7]

As part of the project organisation, the following risks were outlined in the planning phase. Their mitigation strategies were also defined as a guideline, so that a situation does not take us by surprise:

- Restriction to work from home
- Getting off track
- Supervisor unavailable
- Technology gets outdated
- Hardware failure

2.6 Mitigation Strategies

The mitigation strategy was in place in case of one of the risks mentioned takes place or how to avoid the mentioned risk. Following are the strategies in relation to different risks mentioned:

Restriction to work from home:

In case for some reason we are restricted to work from home. I have had previous experience with working from home so it should not be an issue about work management. The effect of this risk would be, it would be harder to get advice from the supervisor and other colleagues. So to avoid this there can be regular planned online meetings where questions about issues can be asked and the availability of different software developers is to be taken into account for guidance.

Getting off track:

In the case of getting off track from the project planning the completion of the project would be risked. To handle this risk there can be formulation of a new revised plan and new prioritisation of the requirements, based on the available time constraint.

Supervisor unavailable:

In case the supervisor is unavailable, the guidance about the project during that time can be received from another senior software engineer until the supervisor is back to work.

Technology gets outdated:

To avoid this risk the technology researched for this project is made sure to be up to date at the moment. And then if something gets outdated, a research for a replacing technology can be conducted.

Hardware failure:

If there is a hardware failure the supervisor is contacted to find an alternative hardware option to continue the project.

2.7 Quality Assurance

Quality Assurance is a means to monitor the process and methods used in a project, to assure good quality of work on a project. Commonly QA includes software testing, design testing, peer review and code authorisation protocol etc.

Testing was a part of the software development process for this project. The User Interface and architecture designs were signed off through peer assessment. The code uses Linters and build tests. Finally, every two weeks the supervisor reviewed the code being written, and gave guidance on best practices. This assured a good quality on the coding/development side.

The limitations and risks mentioned earlier were taken into account while planning and conducting the work. Additionally the project management was done in an agile way. This assured a good quality in management of the project.

2.8 Definition of done

Small tasks were determined based on the task stories in the backlog; if they match the criteria for completion, they are considered completed. When a major milestone or feature that was agreed upon with the stakeholders was completed, the working demo was delivered to the development team. Following the demo, a feedback was taken to determine whether it is complete or not, and how it can be improved. These methods served to stay in the scope of the project and as a definition of done, allowing to move on to the next significant task after completion.

3. Research

This chapter aims at reporting the research findings from the research phase of the project. The research questions were defined during the planning of the project, in chapter 1.4 but the actual research was carried after the planning. Furthermore there was a research plan in chapter 1.6 mentioning the research methods to be used as a guide to answer each subquestion. In the subsections of this chapter the subquestions are answered. The main question is answered in the conclusion of this report as the main question's answer consists of an implementation of the whole project.

Main Research Question

- **How can we centralise the management of translations for many projects?**

Subquestions:

- What are the available existing solutions and how do they compare to each other?
- How can we develop our own platform?
- How can our solution be integrated into existing and new projects?
- How is it possible to integrate automated translations into our own platform?

3.1 What are the available existing solutions and how do they compare to each other?

Methods:

Library research

- Literature study
- Available product analysis
- Best good and bad practices

Method	Description
Library	Done by studying the available work done that can help us in our research.

Table 3.1.1: DOT framework

If we consider the workflow at ProfitFlow, at the moment there is no existing solution. Currently the text is manually translated and manually written to the localisation files of the projects. As the requirement is to come up with a platform that helps manage translations, we would first look at existing solutions in the market.

There are many existing solutions in the market which are similar to a solution to this problem. The similar available products were analysed, but they did not meet the specific requirements of the issues at hand. Some of the platform that were analysed are: [\[8,9,10\]](#)

Lokalise

Lokalise is a multinational firm established in Riga, Latvia. It's a software platform for managing technical product translations and localisation. It's a platform that runs on the cloud. The software makes it easier to translate mobile apps, games, and websites

Locize

Locize is an online web-based Translation management system. Owned by inweso GmbH based in Switzerland found in 2012. It is also used for management of translations for tech products, helping the development process.

POEditor

POEditor is another competitor in the localisation platform market. It has many big brands as their users. It is written in the C++ language. And also facilitates in handling translations for different software products. It is very famous in the WordPress development community.

These platforms were analysed in detail in the advise report which can be found in the [Appendix III](#).

As these mentioned platforms are hosted by well known companies availability, performance, and security are well taken care of. When it comes to scalability it should also not be an issue to scale with these platforms to a reasonable extent as all of these platform have support for many projects at their high tier plan.

But scalability comes at a cost with these platforms. Although it can be seen in Table 3.1.2 that it becomes cheaper per word with every level of upgrade to the payment plan. The issue here is ProfitFlow as a company needs to scale very fast as there are regularly new clients with new projects, which are long-term managed by the company itself. So as the number of

clients and the projects starts increasing it can reach to a level where managing all projects under one payment plan is not reasonable from the cost and the management perspective.

As there can be a situation where a client wants to change the company that manages the project, in that case the management of translations for that project would also need to be handed over with the product. And removing the project from the platform would cost the company. Because the income from the client for the translation would stop but ProfitFlow would still be paying the fee for the monthly plan for the platform. Apart from that moving the project from one account to another is not very easy to manage. Another option having a different account on the platform for every customer makes more sense. But that increases the costs for each customer, as there needs to be a separate plan.

Another thing to consider during this advice was about the way the keys are stored. As a feature there could be a possibility to have some common translation keys for free e.g sign-in. So the common translation keys could be used by any user without paying for them. This can be a competitive advantage and help in management of common keys. For this purpose in the architectural design use of templates was included.

In the advise report ([Appendix III](#)) considering the cost and client management aspect, the conclusion drawn is development of a new platform. In addition the analysis gives us insights on some common features implemented by these existing platforms which are a market requirement. Also gives an idea on how to best price the translation platform services if the project is developed into a Software as a service platform.

	Integration	Scalability	Pricing Model	Pricing
Lokalise	Different mobile apps, web-apps, games	Limited number of allowed developers	Monthly fixed price for limited number of words & limited number of developers	\$585 30,000 words 15 developers
Locize	i18next library	Costs increase with higher usage	Price per word, per modification & per download	\$0.004/word for 10,000 words + \$0.02/modification for 1000 + \$0.000075/download if downloads > 1 million
POEditor	IOS, Android, Angular, React and many more	Limited number of words	Monthly fixed price for limited number of words	\$200 100,000 words

Table 3.1.2: Comparison of existing platforms

3.2 How can we develop our own platform?

Methods:

Field research

- Problem Analysis
- Survey
- Interview
- Explore user requirements

Method	Description
Field	Done to better understand the situation of a problem and the stakeholders requirements for a proposed solution.
Workshop	Done to explore opportunities and how things could work. Prototyping, designing and co-creation.

Workshop

- Brainstorming
- IT architecture sketching
- Code implementation & Prototyping

Table 3.2.1: DOT framework

3.2.1 Problem Analysis:

ProfitFlow is in charge of its clients' codebases. Scalability with the existing platforms investigated in the market comes at a cost. Aside from that, it is difficult to administer those platforms if a client wishes to delegate the project to another company; to do so, a membership for each distinct project on those platform is required, which comes at an additional expense.

So, in order to tackle this problem and make the company's development process more effective, I concluded that creating a new platform would be the best option. The next steps were to conduct a survey, interview, and gather requirements in order to have a better grasp of the company's and developers' needs. The survey, interview, and requirement analysis reports can be found in the following sub-chapters.

3.2.2 Survey

As part of the field research a survey was conducted on all the software developers working for ProfitFlow. The survey was designed from the questions that came up after brainstorming on the problem analysis.

First part of the survey was to get an understanding of the frameworks, libraries and technologies currently being used at the company. The findings were that mostly React or NextJS frameworks were used for front-end development. The library mostly used for localisation was i18next. It was found that most of the projects used localisation.

Next part was some questions about how the current process is and how satisfied they are with it. The developers were not satisfied with the current process as a lot of manual and repetitive work had to be done on daily basis.

Finally I asked for features they wanted. And auto translate and CLI integration were the favourites in the wish-list of features.

The full survey results can be found in the [Appendix II](#) of this report.

3.2.3 Interview

This is an interview with the graduation supervisor, who is also the company's Technical Director and a stakeholder in this project. This interview was created using the insights from the survey, which the developers completed as stakeholders. The interview was conducted to obtain a more in-depth analysis after the survey results.

What are the type of projects that need translations?

- Maatwerksoftware (custom build software for companies, such as planning tools etc)
- Software-as-a-Services

How is the translation process carried out currently?

The developers do live translations during the development. No magic strings are allowed. There is no systematic approach currently documented.

How do you think we can improve the process without a new platform?

By documenting a workflow that has to be adhered to.

Do you think a platform could make the the translation process more efficient?

Yes and especially more manageable. Currently all translations are stored in JSON files in the source repository. In order to change translations the repo has to be cloned and edited. Using a platform we can automate this process during build and gather all latest translations.

Will this platform only be used by the ProfitFlow developers or also external users?

In the beginning stages only ProfitFlow developers will use the tool. In the future other ProfitFlow employees (Business / sales) would like to edit translations as well. Meaning there should be a role system.

Also we might consider SaaSifying the whole platform in the far future.

What qualities in the workflow do you think are needed for the developers to be motivated to use such a platform regularly?

Automatically retrieving the translation files during build for production

Automatically retrieving the translation files using APIs during development (for real-time editing during dev)

What are the functional requirements for the platform?

We are going to have to do this one together and plan a couple hours

What are the non-functional requirements for the platform?

We are going to have to do this one together and plan a couple hours

3.2.4 Requirements

After using the different research methods mentioned for this research question, like survey, talking to stakeholders and supervisor, I was able to come up with the software requirements. The main requirements were following in a MoSCoW format: [11]

Functional requirements:

Functional requirements are the requirements related to the business functionality of the project. For example the actions the user should be capable of performing while using the platform.

#	Requirement	MosCow	Status
F#1	The platform must be capable of creating / editing projects	Must	Done
F#2	The platform must provide dashboarding per project	Must	Done
F#3	The platform must be capable of creating / editing groups	Must	Done
F#4	The platform must be capable of adding groups to projects	Must	Done
F#5	The platform must be capable of adding templates in a project	Must	Done
F#6	The platform must be capable of creating / editing languages	Must	Done
F#7	The platform must be capable of adding groups to projects	Must	Done
F#8	The platform must be capable of creating / editing key/value translations in a project and/or module	Must	Done
F#9	The platform must be capable of creating/editing users	Must	Done
F#10	The platform must contain user roles	Must	Done
F#11	The platform must be capable of authenticating users	Must	Done
F#12	The platform must provide a history of any key/value or project (changes)	Must	Canceled
F#13	The platform must be capable of approving new translations (done by either the customer or project/operational manager) for any project	Must	Canceled
F#14	The platform should be capable of creating/editing translators (that have limited rights)	Should	Done
F#15	The platform should provide dashboarding on global level	Should	Done
F#16	The platform could identify reoccurring key/value pairs	Could	Done
F#17	The platform could be capable of translating automatically using any ML API (Google, etc..)	Could	Done
F#18	The platform could include "free" common words (gotten from the value)	Could	Done

Table 3.2.2: Functional requirements

UX requirements:

UX requirements are requirements which are only related to visual and user experience of the platform these do not affect the functionality but only effect the user experience.

#	Requirement	MosCow	Status
UX#1	The platform must be translated in English and Dutch	Must	Done
UX#2	The platform must provide dark mode	Must	Done
UX#3	The platform must be customizable using different colors	Must	Done
UX#4	A project must include details such as customer details, (optional) logo and default language	Must	Done

Table 3.2.3: UX requirements

Non-functional requirements:

Non-functional requirements are the requirements related to the architecture of the project. They define the technologies desired by the company and features which do not affect the functionality of the project

#	Requirement	MosCow	Status
NF#1	The platform should be deployed using the JAMStack	Must	Not yet
NF#2	The platform must be secured using JWT	Must	Done
NF#3	The platform must be scalable	Must	Done
NF#4	The platform must be using serverless technology	Must	Done
NF#5	The platform should be hosted at Netlify	Must	Done
NF#6	The platform must be designed using Figma	Must	Done
NF#7	The platform must use Hasura for backend	Must	Done
NF#8	The platform must audit all changes done to the database (https://hasura.io/docs/latest/graphql/core/guides/auditing-tables/)	Must	Done
NF#9	The platform should be documented fully	Should	Done
NF#10	The platform should be manually tested, including a (small) test report	Should	Done
NF#11	The platform should be created in NextJS	Should	Done
NF#12	The platform should use PostgreSQL	Should	Done
NF#13	The platform could contain automated testing	Could	Not yet
NF#14	The platform could be regularly backed up	Could	Canceled

Table 3.2.4: Non-Functional requirements

Integration requirements:

Finally the integration requirements were defined as the requirements that are related to the integration of the project with other technologies.

#	Requirement	MosCow	Status
I#1	The platform must be capable of communicating the translations to the project during build time	Must	Done
I#2	The platform could be capable of communicating the translations to the project during compile time	Could	Done
I#3	The platform could automatically initiate a build on staging / production	Could	Done
I#4	The platform could make use of CDN to provide translations to the applications	Could	Not yet
I#5	The platform could integrate with Slack for updates	Could	Canceled
I#6	The platform could integrate with Wordpress	Could	Canceled
I#7	The platform could integrate with Strapi	Could	Canceled
I#8	The platform won't have a vscode plugin that retrieves all keys in a project and autocompletes in the IDE this time	Wont	

Table 3.2.5: Integration requirements

Conclusion:

In conclusion these requirements were taken into account while creating the designs in chapter 4 and also during sprint planning for realisation. Most of the requirements are met by the end of the project. Some were cancelled during the project in consultation with stakeholders. F#12 and F#13 were not needed anymore and NF#14, I#5 - I#8 were out of the current scope of the project. As for all remaining requirements marked as “Not yet” are NF#1 in progress for the final handover stage.

3.3 How can our solution be integrated into existing and new projects?

Methods:

Field research

- Explore user requirements
- Task Analysis

Workshop

- IT architecture sketching
- Code implementation & Prototyping

Lab research

- Usability testing

Method	Description
Field	Done to better understand the situation of a problem and the stakeholders requirements for a proposed solution.
Lab	Done to test parts of the product, to be sure things work the intended way.
Workshop	Done to explore opportunities and how things could work. Prototyping, designing and co-creation.

Table 3.3.1: DOT framework

Here by integration it is meant that the keys and values from the platform are downloaded to the development files of a project. This was required, because this made the development process easier which was the main goal of this project. To Integrate the Translations into Local Project's we needed to make some connection between the platform and the local projects.

First let's look how the translations were stored previously in the projects:

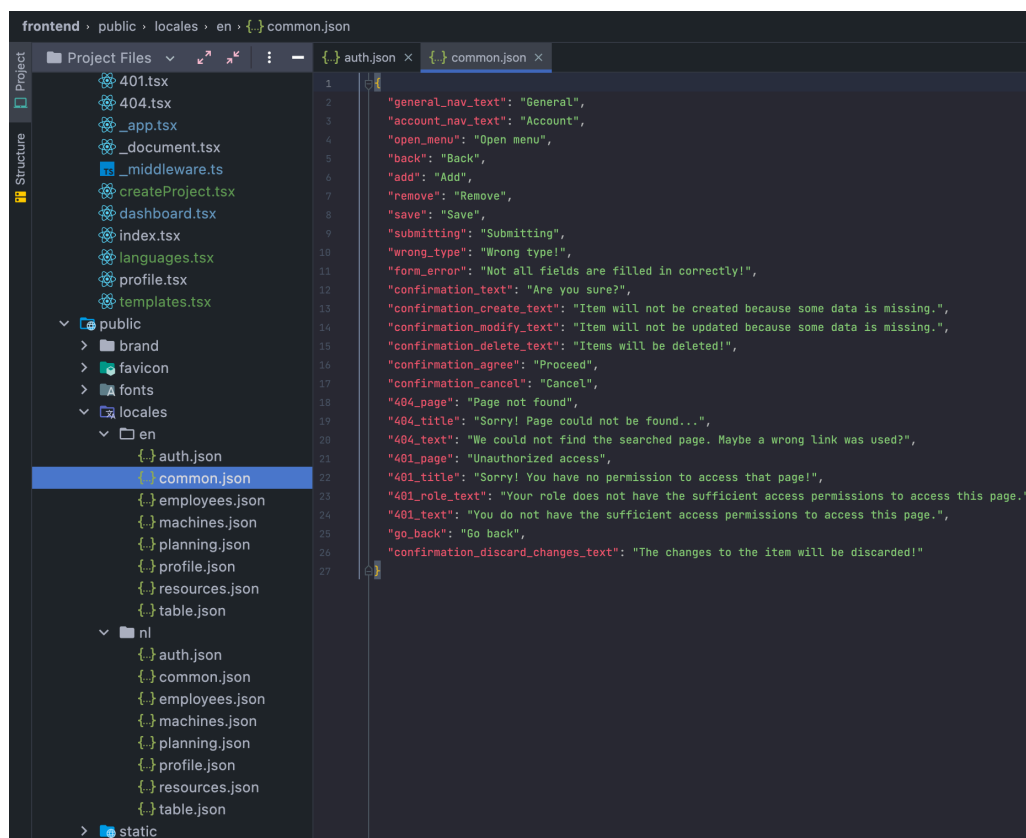


Fig 3.3.1: locales folder structure

The translations were originally saved in the locales directory, with a directory for each available language, as seen in the screenshot. The modules are stored in JSON format in each language directory. These JSON modules contain the key and translation pairs. Translations for keys are retrieved from modules from the relative language directory when a language is selected on the UI. As a result, instead of a hardcoded text, the module and key are used in the Javascript and html during the development process. Libraries such as i18next assist in obtaining translation values.

Because the currently used i18n library has the above shown structure. Taking scalability into account, the purpose was to download data from the new platform in the same format and file structure. This would make integration process of existing company projects effortless.

There were two possible ways to download the data:

- Create a service that gets all the data from the backend puts it into files. Then over the network the files are sent to the client who requested. After retrieving the files they can be stored in the locale folder.
- Create an NPM library which can be installed on the client. Whenever needed use a CLI command, to retrieve the data inside the library, and create the files locally to store in locale folder.

After some research and discussion with stakeholders I decided to go with the second option as it made more sense with performance and user experience. With the first option we would have to download written files, which is slower and with more steps. With the second option we download the data, and create the files locally.

Initialise

The next step was learning how to create an NPM library. To create an NPM library we start by setting up the package.json file in a new directory. Inside the package.json there are some required fields like package name, version, license info which need to be set. The name of the package was set to locflow-downloader. [12]

Write the function

The most innovative part was to write the index.js file. This file had the actual function that loads the data from the backend, and writes into local files. The data is loaded using the projectID. The projectID is provided to the function through environment variables in .env files. This is because the library needs to be reusable with any project. To load the data from GraphQL backend I use the Apollo library. Then I use the fs library provided by node.js to create and write files using the loaded data. [13]

Publish Library

Finally we want to publish or create a package for the library we have written. As I wanted to locally develop this library for the prototype I do not opt to publish it to the NPM registry. Instead I use NPM pack command to bundle the package locally. After this a package is created locally which can be Installed using NPM install. [14]

Testing

After creating the library, I create a new nodeJS project to test the NPM library. First to initialise the project I use NPM install <path to the local package>. Install the package and set up the .env variables. Now the function can run using npx locflow-downloader and it downloads the translation files into the locale folder, as required. [15]

We can use an NPM package to retrieve translations into our projects so this fulfils our goal for integration.

The environment variables required in a project where the library is used are:

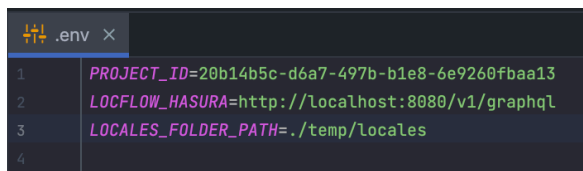


Fig 3.3.2: .env variables

The function in this NPM library that fetches the data and writes to local files is shown in the Fig 3.3.3. In simple words the steps it takes are:

- Fetches the data from the backend using the project_id provided in the env variables
- Creates the language directories based on supported languages at the platform
- Adds each Group to all language folders
- Adds keys with the relevant translation to the Groups
- If the -replace flag was set on the npx command it replaces the group completely, removing the local changes

```
const project = res.data.project;
if (project) {
  console.log("Successfully Downloaded")
  project.languages.forEach((language, project) => {
    const iso = language.iso
    const languageFolderPath = process.env.LOCALES_FOLDER_PATH + '/' + iso

    if (!fs.existsSync(languageFolderPath)) {
      fs.mkdirSync(languageFolderPath, {recursive: true});
    }

    project.groups.forEach((group) => {
      const groupFileName = languageFolderPath + '/' + group.name.replace(/s/g, "") + '.json'
      const readExistingFile = fs.existsSync(groupFileName) && !process.argv.includes("--replace")

      try {
        const groupJSON = JSON.parse(readExistingFile);
        group["group_keys"].forEach((key) => {
          const keyToSave = key.key
          const translation = key["translations"].find(translation => translation.language === language.iso)
          if (translation) groupJSON[keyToSave] = translation["translation"]
        })

        const data = JSON.stringify(groupJSON, null, 2);
        fs.promises.writeFile(groupFileName, data)
          .then(() => console.log("Written to [" + iso + "] -> " + groupFileName))
          .catch(console.error)
      } catch (e) {
        console.log(groupFileName, e)
      }
    })
  })
}
```

Fig 3.3.3: Function that fetches the data and writes to locales

Apart from this research, the design chapter has an architecture design on how the components are put together, and how they work in relation to each other. The advise section gives in detail advise on how we can implement an efficient workflow for the development part of translation.

3.4 How is it possible to integrate Google Translate into our own platform?

Methods:

Library research

- Literature study
- Best good and bad practices

Workshop

- Code implementation & Prototyping

Method	Description
Library	Done by studying the available work done that can help us in our research.
Workshop	Done to explore opportunities and how things could work. Prototyping, designing and co-creation.

Table 3.4.1: DOT framework

To Integrate Google translate feature into our platform we would need to use the Google Translate API. For which most of the documentation is provided on the Google Cloud platform. The API usage is a paid feature but the pricing is not too much so the company stakeholders were willing to get research on the Translate API. In this research the Translate API's integration, guidelines and pricing are discussed.

Cloud Translation offers two editions:

- Cloud Translation - Basic (v2)
- Cloud Translation - Advanced (v3)

Both editions support language detection and translation. They also use the same Google pre-trained model to translate content. Cloud Translation - Advanced includes features such as batch requests, AutoML custom models, and glossaries. As we only need the translation feature we will be using the Basic version.

Even though the Basic version has the language detection feature but we will be only using the translate feature because according to my design we will always be translating from English language. As per pricing the first 500,000 characters are translated for free the next are charged \$20 per million characters.

First we need a project on the Google Cloud, with the Cloud Translation API enabled and the project credentials to make authenticated calls and only then we can use Cloud Translation.

From the project we can get the private key file, put it in our project directory and then link the key with an environment variable.

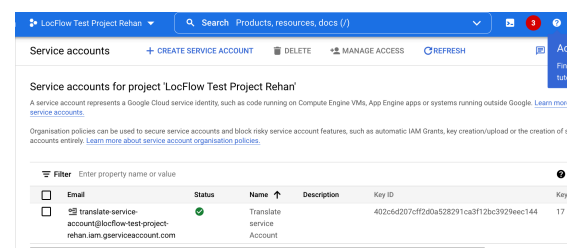


Fig 3.4.1: Google Cloud Platform

To enable the API we need a billing Account. For safety we can also set a usage quota in the Google Cloud. So that the costs don't go very high if a lot of bad or accidental requests are made. Then after setting up a billing account for the project in Google Cloud we can enable the API. Now that the API is enabled we can make REST API calls to the API to get our translations.

REST API:

POST <https://translation.googleapis.com/language/translate/v2>

Parameters:

Parameters	
q	<p>string</p> <p><i>Required</i> The input text to translate. Provide an array of strings to translate multiple phrases. The maximum number of strings is 128.</p>
target	<p>string</p> <p><i>Required</i> The language to use for translation of the input text, set to one of the language codes listed in Language Support.</p>
format	<p>string</p> <p>The format of the source text, in either HTML (default) or plain-text. A value of <code>html</code> indicates HTML and a value of <code>text</code> indicates plain-text.</p>
source	<p>string</p> <p>The language of the source text, set to one of the language codes listed in Language Support. If the source language is not specified, the API will attempt to detect the source language automatically and return it within the response.</p>
model	<p>string</p> <p>The translation model. Cloud Translation - Basic offers only the <code>nmt</code> Neural Machine Translation (NMT) model.</p> <p>If the model is <code>base</code>, the request is translated by using the NMT model.</p>
key	<p>string</p> <p>A valid API key to handle requests for this API. If you are using OAuth 2.0 service account credentials (recommended), do not supply this parameter.</p>

Fig 3.4.2: Google API request parameters

We can send the API requests to the above mentioned API with the parameters in the figure to get a response like the following:

```
{
  "data": {
    object(TranslateTextResponseList)
  },
}
TranslateTextResponseList : {
  "translations": [
    {
      "detectedSourceLanguage": string,
      "model": string,
      "translatedText": string,
    }
  ],
}
```

After getting the correct response we can display it in our FrontEnd Application, this will be explained more in the Realisation chapter. [\[16\]](#)

TranslationService:

Setting up the account on google cloud platform was the first step, next was creating a TranslateService that is using the Translation API and sending the data back to the front-end.

From the initial research I had thought that we can use the translation API directly on the front-end client. But this was a blind spot in this research after some code implementation it was found that for security reasons Google does not recommend making API calls directly from the front-end. To authenticate Google uses a private key, if we put the private key in the front-end application it becomes vulnerable to attacks. So to avoid this security threat the best practice is to create a nodeJS TranslateService that stores the private key. And make api requests to the Google API when ever the front-end client makes a request to the TranslateService. [17]

Now when a user clicks the translate button on a cell it sends a request to the translation service with the english string and the target language.

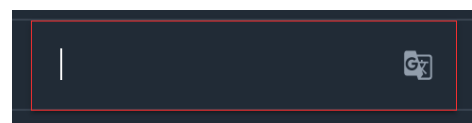


Fig 3.4.3: Translation button inside a cell

```
async function translateText(text, target) {
  // Translates the text into the target language. "text" can be a string for
  // translating a single piece of text, or an array of strings for translating
  // multiple texts.
  let [translations] = await translate.translate(text, target);
  translations = Array.isArray(translations) ? translations : [translations];
  // console.log('Translations:');
  // if (translations.length <= 1) console.log(text,"=>",translations[0])
  // else {
  //   translations.forEach((translation, i) => {
  //     console.log(`${text[i]} => (${target}) ${translation}`);
  //   });
  // }
  return translations
}

// add router in express app
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({extended: true}));
app.use("/", router);

router.post( path: "/translate",  handlers: (req : Request<P, ResBody, ReqQuery, Locals>
  translateText(req.body.text, req.body.target)
  .then(translationRes => {
    res.send( body: {translations: translationRes})
  }).catch(ex => res.status( code: 500).json(ex))
));

const hostname = 'localhost';
const port = 4200;
app.listen(port, hostname: () => {
  console.log("Started on PORT " + port);
})
```

Fig 3.4.4: TranslateService

The TranslateService upon receiving a request at api/translate . Sends a request to the Google Translate API. When it receives a response with translated string it send back the response to the client who made the request.

The target languages supported by the Google service and their relative iso codes are specified in Google docs at [18]

Language	ISO-639-1 Code
Afrikaans	af
Albanian	sq
Amharic	am
Arabic	ar
Armenian	hy
Azerbaijani	az
Basque	eu
Belarusian	be
Bengali	bn
Bosnian	bs
Bulgarian	bg
Catalan	ca
Cebuano	ceb (ISO-639-2)

Fig 3.4.5: Language Support Google

4. Design

4.1 Architecture Design

The Architecture design was created after the research phase. And it was regularly updated during the development sprints.

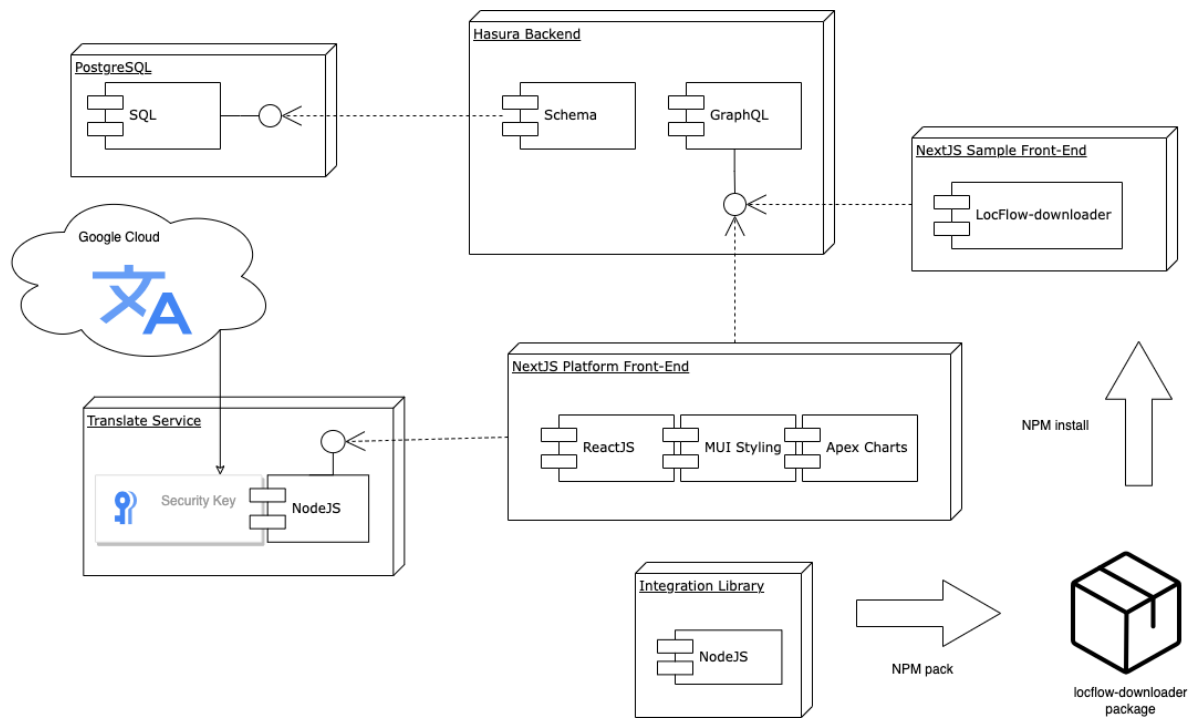


Diagram 4.1: Architecture diagram

This project's architecture is made up of several parts, the majority of which are linked to one another. I use a PostgreSQL database that is linked to the Hasura Backend.

The platform's front-end communicates with the backend to query or manipulate data. On the other hand, it connects to our Translate Service to get translations. The Google Translate service connects to the Google API using the security key to request and fetch translations. To aid development, the front-end also makes use of libraries such as Apex charts and MUI styling.

Finally, we have the integration library, which generates an NPM package. The NPM package can be installed on any project. After installing, it can get the data by querying from the backend.

4.2 Business Design

The Business design was created based on the stakeholders requirements. There were many changes, after reviews on the whole project, and when some main requirement changed. The requirements from chapter 3.2.4 required to have create, read update and delete functionality on users, projects, groups, keys and translations. This represents the final and best version of the design.

There are these main components:

- Users
- Projects
- Groups
- Templates
- Keys
- Translations
- Languages

Users can have Roles, the project and Platform features access is based on these roles.:

1. Administrator
2. Developer
3. Translator

Projects can have users assigned to them. And one project can have many groups. These groups contain keys with their translations.

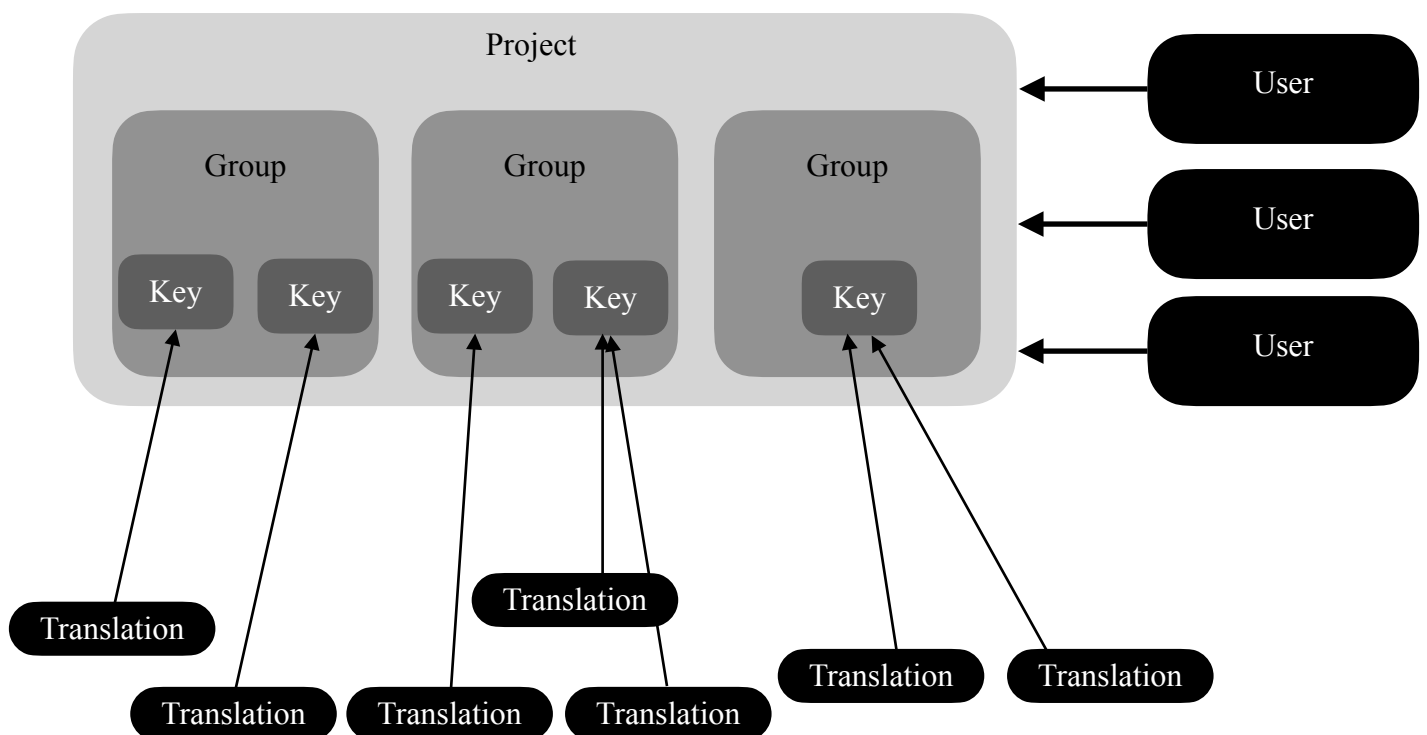


Diagram 4.2: Business Design

4.3 Database design

As we can see from the previous section about Business Design we needed 6 main tables:

- user
- project
- group
- key
- translation
- language

To connect these tables we needed some extra join tables. A user_role table was needed to define roles for users with different levels of access.

A project can have users so we have join table between project and user table. And similarly are the project_group, group_key tables. Templates from the business design however are groups in the database, with a boolean called template, and a value: true. Because projects stored the languages and templates are not related to any project, the templates have a template_language table to store the supported language for each template.

There were a couple of design iterations to the db because of change of requirements or implementation. The summary of database design evolution can be found on [Appendix IX](#). Diagram 4.3 shows the final version of db schema design :

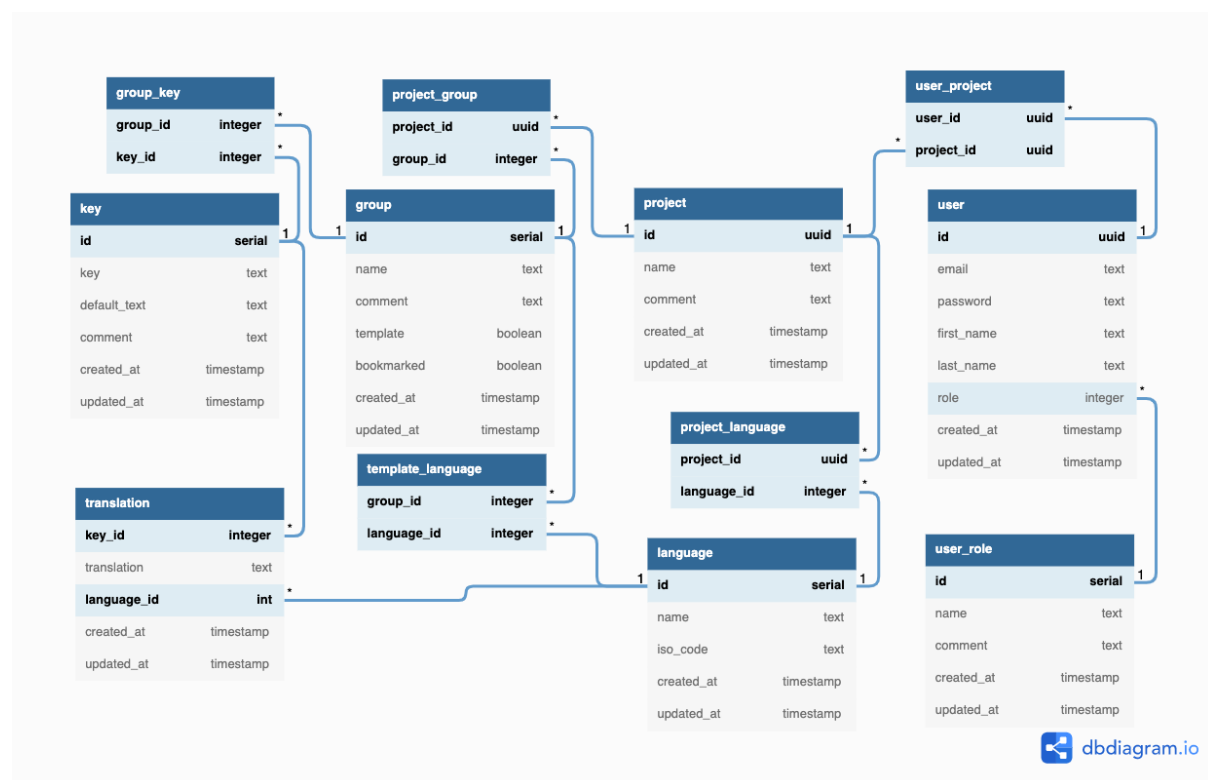


Diagram 4.3: DB schema final version

4.4 Frontend Mockup

This was the initial mockup designed to have an idea of how the platform should be looking:

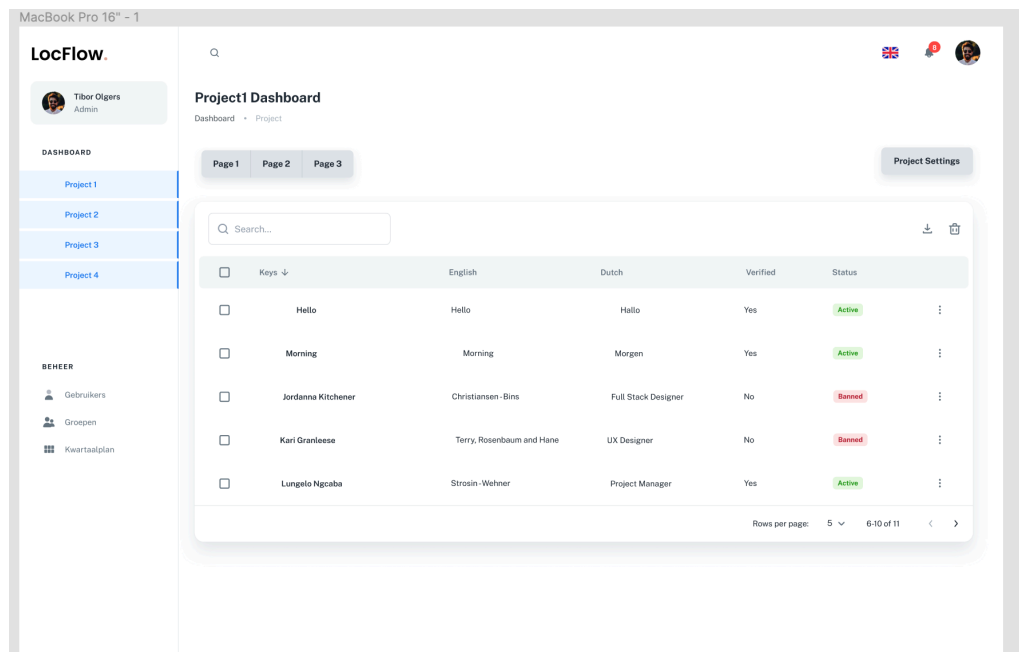


Fig 4.1: Frontend Mockup

The frontend, however had several alterations throughout the development. During several demos, developers advised different changes. For example we remove the project list from the sidebar because we would potentially have a large number of projects. This is how the frontend looks now:

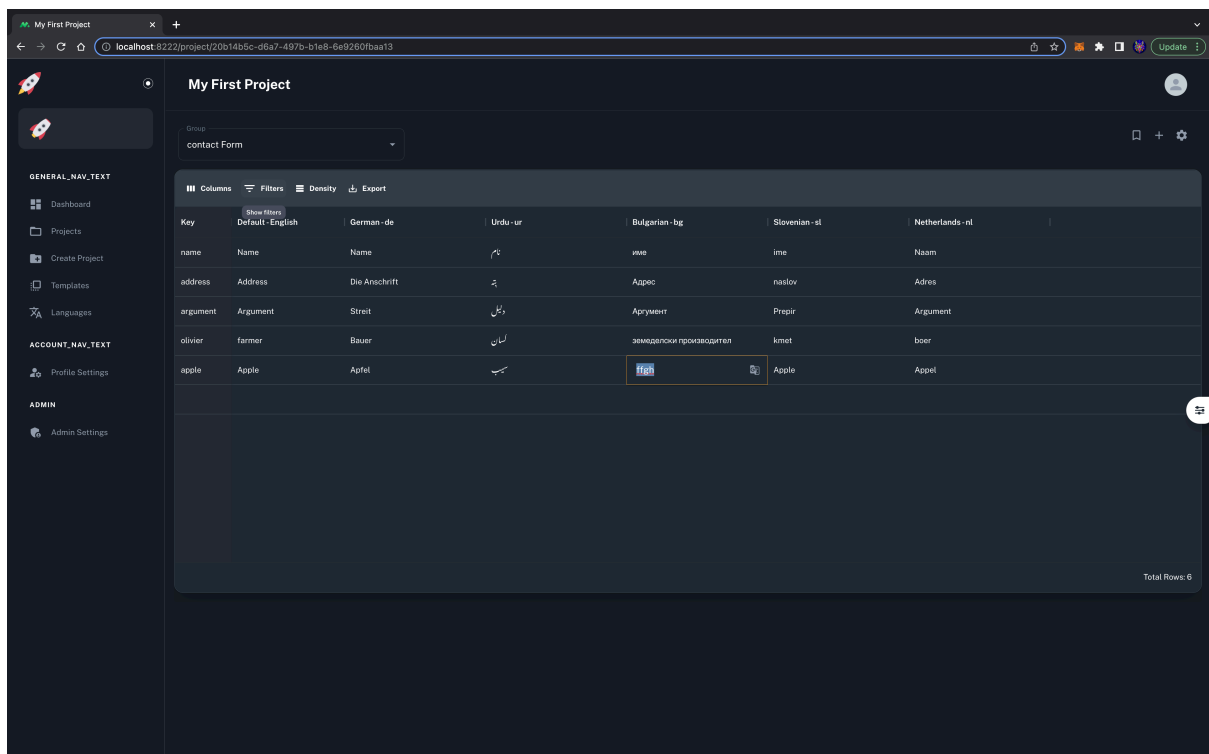


Fig 4.2: front-end screenshot

4.5 Styling

This project's styling was done with the help of a Styling Library. Following a proposal and discussion with the company's Technical Director, the decision to use a styling library was made. The benefit of using a library is that it makes the styling more consistent and the development more efficient.

The library used was MUI from mui.com. It is a react-supported npm library. Many of MUI's supported components can be reused. However, for specific use cases, the components provided need to be customised and tailored to my requirements. For example, while using the data grid from MUI, I needed to add the translate button to the cells. I modified the code for the cells to include a button inside a cell when it is in edit mode. Furthermore, I had to create a custom export format for when someone wanted to download the data from the data grid in JSON format directly. [19]

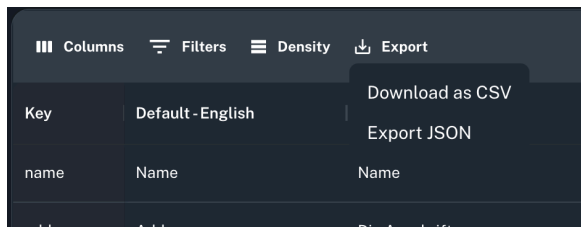


Fig 4.3: JSON export option

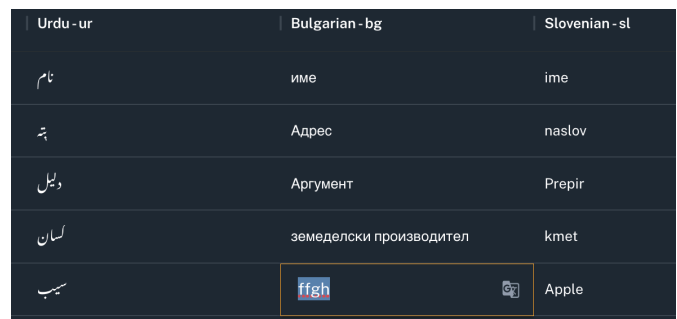


Fig 4.4: translate button in a cell

I also made use of the Apex Charts library to display project statistics in chart form on the dashboard. [20]

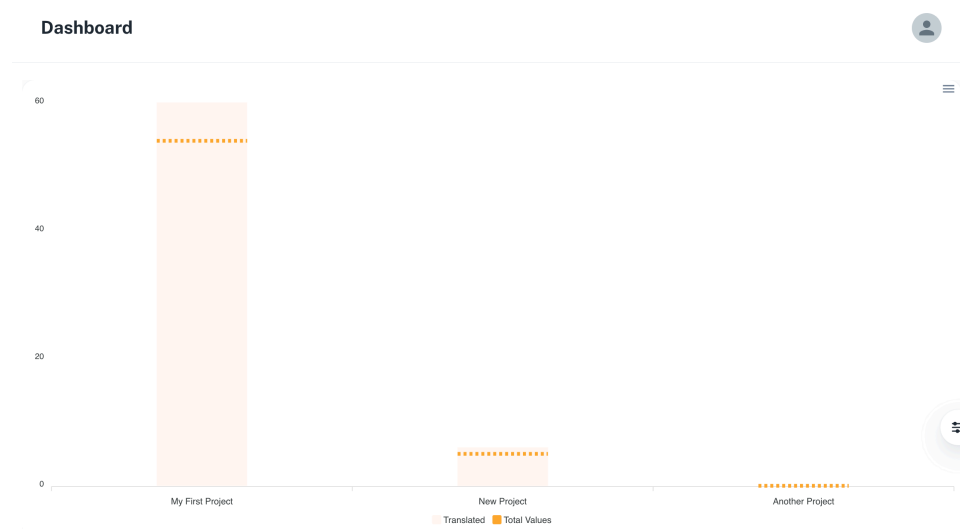


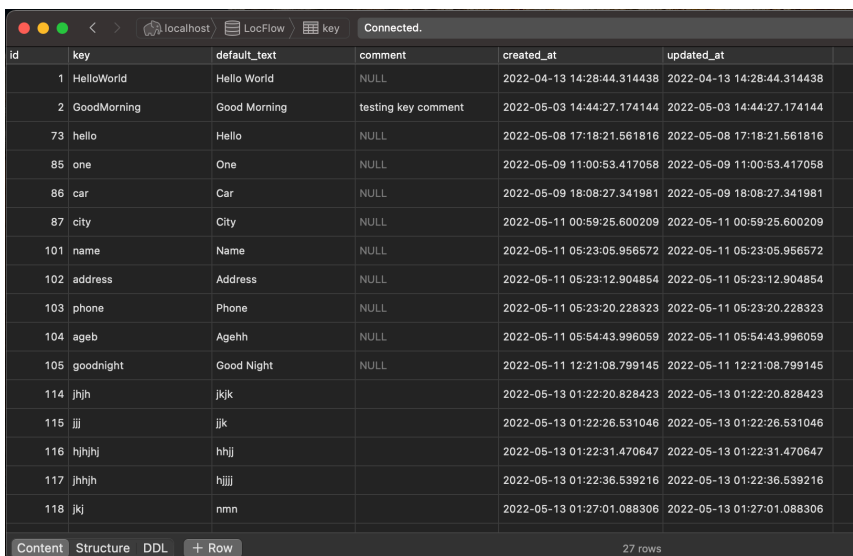
Fig 4.5: Chart for statistics on dashboard

5. Realisation

5.1 Database:

First step in realisation was to setup a Postgres database local environment on the local machine. Next, using the database design mentioned previously in chapter 4.3, SQL code is written to create the SQL tables and their relations.

PgAdmin4 and Postico were used interchangeably to manipulate data and make any required changes inside the database during development.



id	key	default_text	comment	created_at	updated_at
1	HelloWorld	Hello World	NULL	2022-04-13 14:28:44.314438	2022-04-13 14:28:44.314438
2	GoodMorning	Good Morning	testing key comment	2022-05-03 14:44:27.174144	2022-05-03 14:44:27.174144
73	hello	Hello	NULL	2022-05-08 17:18:21.561816	2022-05-08 17:18:21.561816
85	one	One	NULL	2022-05-09 11:00:53.417058	2022-05-09 11:00:53.417058
86	car	Car	NULL	2022-05-09 18:08:27.341981	2022-05-09 18:08:27.341981
87	city	City	NULL	2022-05-11 00:59:25.600209	2022-05-11 00:59:25.600209
101	name	Name	NULL	2022-05-11 05:23:05.956572	2022-05-11 05:23:05.956572
102	address	Address	NULL	2022-05-11 05:23:12.904854	2022-05-11 05:23:12.904854
103	phone	Phone	NULL	2022-05-11 05:23:20.228323	2022-05-11 05:23:20.228323
104	ageb	Agehh	NULL	2022-05-11 05:54:43.996059	2022-05-11 05:54:43.996059
105	goodnight	Good Night	NULL	2022-05-11 12:21:08.799145	2022-05-11 12:21:08.799145
114	jhjh	jkjk		2022-05-13 01:22:20.828423	2022-05-13 01:22:20.828423
115	jjj	jjk		2022-05-13 01:22:26.531046	2022-05-13 01:22:26.531046
116	hhjhj	hhjj		2022-05-13 01:22:31.470647	2022-05-13 01:22:31.470647
117	jhjhj	hhjjj		2022-05-13 01:22:36.539216	2022-05-13 01:22:36.539216
118	jkj	nmn		2022-05-13 01:27:01.088306	2022-05-13 01:27:01.088306

Fig 5.1: Postico

```
1 CREATE TABLE "user_role" (  
2   "id" serial PRIMARY KEY,  
3   "name" text UNIQUE NOT NULL,  
4   "comment" text,  
5   "created_at" timestamp NOT NULL DEFAULT (now()),  
6   "updated_at" timestamp NOT NULL DEFAULT (now())  
7 );  
8  
9 CREATE TABLE "user" (  
10  "id" uuid PRIMARY KEY DEFAULT (gen_random_uuid()),  
11  "email" text NOT NULL,  
12  "password" text NOT NULL,  
13  "first_name" text NOT NULL,  
14  "last_name" text NOT NULL,  
15  "role" integer NOT NULL,  
16  "created_at" timestamp NOT NULL DEFAULT (now()),  
17  "updated_at" timestamp NOT NULL DEFAULT (now())  
18 );  
19  
20 CREATE TABLE "user_project" (  
21  "user_id" uuid,  
22  "project_id" uuid,  
23  PRIMARY KEY ("user_id", "project_id")  
24 );  
25  
26 CREATE TABLE "project" (  
27  "id" uuid PRIMARY KEY DEFAULT (gen_random_uuid()),  
28  "name" text UNIQUE NOT NULL,  
29  "comment" text,  
30  "created_at" timestamp NOT NULL DEFAULT (now()),  
31  "updated_at" timestamp NOT NULL DEFAULT (now())  
32 );  
33  
34 CREATE TABLE "group" (  
35  "id" serial PRIMARY KEY,  
36  "name" text NOT NULL,  
37  "comment" text,  
38  "template" boolean NOT NULL,  
39  "created_at" timestamp NOT NULL DEFAULT (now()),  
40  "updated_at" timestamp NOT NULL DEFAULT (now())  
41 );  
42  
43 CREATE TABLE "project_group" (  
44  "project_id" uuid,  
45  "group_id" integer,  
46  PRIMARY KEY ("project_id", "group_id")  
47 );  
48  
49 CREATE TABLE "key" (  
50  "id" serial PRIMARY KEY,  
51  "key" text NOT NULL,  
52  "english_translation" text NOT NULL,  
53  "created_at" timestamp NOT NULL DEFAULT (now()),  
54  "updated_at" timestamp NOT NULL DEFAULT (now())  
55 );  
56  
57 CREATE TABLE "..."
```

Fig 5.2: SQL code

PostgreSQL is an open source, object-relational database system. The choice for PostgreSQL is due to the large increase in performance and SQL standards. Postgres implement MVCC[21], meaning that PostgreSQL can use multiple cores in parallel for creating indexes and partial indexes. Aside from this, PostgreSQL also protects its data at the transaction level, meaning it is less vulnerable by design to data corruption.

The decision to use Postgres for database was made in agreement with supervisor as the supervisor wanted Hasura to be used for backend and Postgres was the most compatible database with Hasura

5.2 Backend:

Hasura was used for the backend, as this was one of the must requirements of the company. To setup local hasura environment I ran a docker-compose -up on the following docker file, this downloaded a hasura image and ran a docker container with hasura connected to the database:

```
docker-compose.yaml
1 version: '3.6'
2 services:
3   graphql-engine:
4     image: hasura/graphql-engine:v2.4.0
5     ports:
6     - "8080:8080"
7     restart: always
8     environment:
9       ## postgres database to store Hasura metadata
10      HASURA_GRAPHQL_METADATA_DATABASE_URL: postgresql://rehan:pass@host.docker.internal:5432/LocFlow
11      ## this env var can be used to add the above postgres database to Hasura as a data source, this c
12      PG_DATABASE_URL: postgresql://rehan:pass@host.docker.internal:5432/LocFlow
13      ## enable the console served by server
14      HASURA_GRAPHQL_ENABLE_CONSOLE: "true" # set to "false" to disable console
15      ## enable debugging mode. It is recommended to disable this in production
16      HASURA_GRAPHQL_DEV_MODE: "true"
17      HASURA_GRAPHQL_ENABLED_LOG_TYPES: startup, http-log, webhook-log, websocket-log, query-log
18      ## uncomment next line to set an admin secret
19      HASURA_GRAPHQL_ADMIN_SECRET: myadminsecretkey
20
```

Fig 5.3: hasura docker-compose file

5.3 Hasura:

Backend development has been massively innovated on by the open-source community of software engineers. Hasura was released with the idea that there is no longer a need to develop a REST (or GraphQL) CRUD backend service from scratch.

Hasura provides the developer's, with a quick and easy way of deploying a full-fetched CRUD based on an existing database. One of the major features that comes with Hasura is an RBAC system that allows the developer to define complex permissions on rows and columns of tables within the database.

One of the biggest advantage of Hasura is it produces a quick GraphQL schema based on the database that it is connected to. So this saves time from creating a graphql schema

Now after connecting the frontend to the endpoint there needed to be made custom functions in the backend. Like a trigger on a value update, to save new update time. After set up we can quickly start making requests.

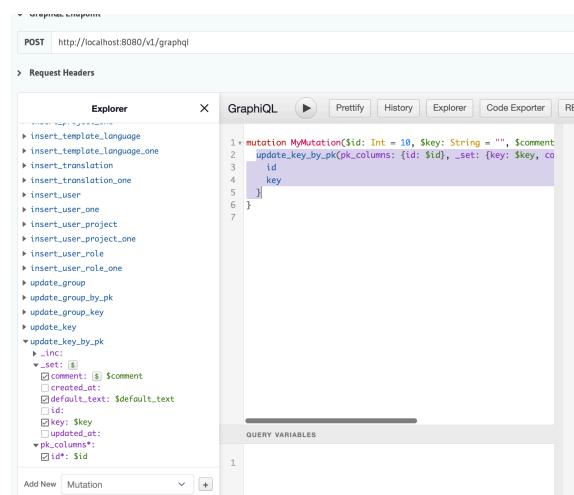
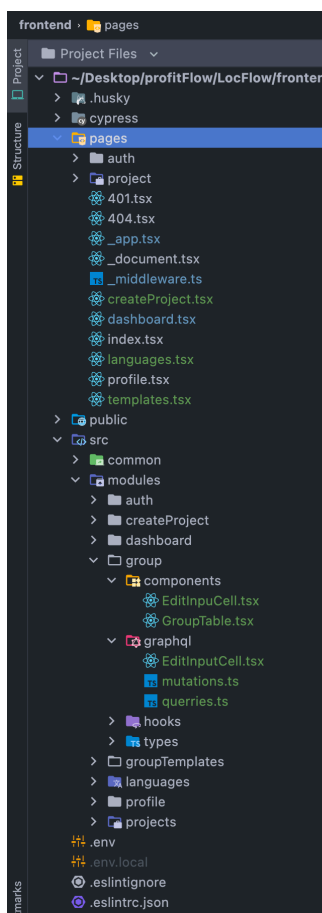


Fig 5.4: hasura API explorer

5.4 Frontend:

Frontend application uses a react based NextJS framework. NextJS is an open-source development framework built on top of Node.js that enables server-side rendering and the generation of static websites in React-based web apps.

A web application framework simplifies, accelerates, and scales the development process. I picked this framework since the company already uses NextJS for the majority of its applications, making it easy to integrate a standalone app with existing web-apps. Apart from that it is one of the frameworks I have not worked with before, thus it was a fresh experience for me.



The frontend is structured into folders of :

- pages
- public
- src

pages:

Contains all the static pages and the NextJS router links these pages to routes based on the page name

public:

Contains all the public assets which are needed at the build time

src:

contains all the modules a module can be used in other modules. A module directory contains the components, their graphql queries/ mutations, hooks and types.

Fig 5.5: file structure

Some of the main modules are group, auth, createProject. To optimise code and use a common component for similar functioning parts, the structure of the project changed over the course of the graduation. For example the GroupTemplates before it was called this was called just Groups which had its own keys and translations, projects also had their own keys and translations. The idea was that different groups could be made reusable with different projects. But later on I decided the projects did not need to hold keys so made it so that only groups can have keys. Projects hold those groups. So in this way the table component which had logic for updating each cell or adding keys became reusable. Now groups and groupTemplates both can use the same table component.

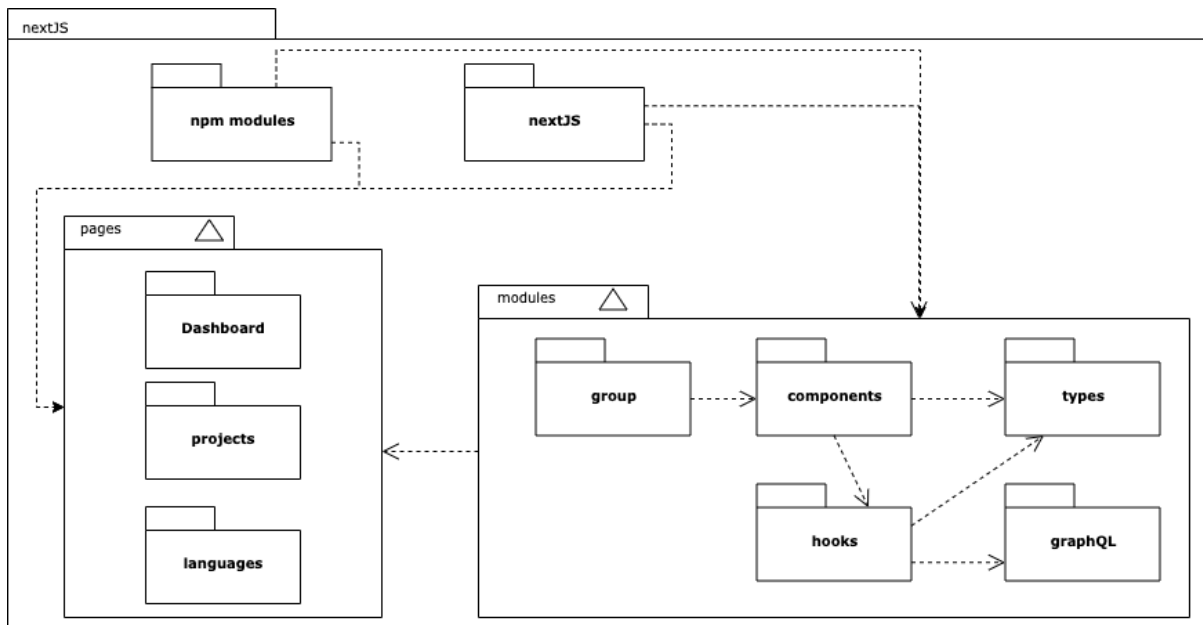


Diagram 5.1: Project structure

Each module contains the following files:

- components
- graphql
- hooks
- types

components:

This folder contains all the .tsx files with the html and typescript for a child component of the parent component. For example the Group contains custom edit cells which are child components of the parent component Group.

graphql:

This folder contains the graph QL queries and mutations. Written in graphql format. It is a good practice to keep the queries and mutations separate from the rest of the code. Because it would also be easy in future to customise the query or mutation if the database/backend design changes.

hooks:

Hooks folder contains typescript files which are responsible for handling the query or mutation. For example if some event like clicking a button was loading some data from the backend. The buttons click handler would call a hook to fetch and load the data to update the components state.

types:

As I am using typescript, defining types for the objects assures a good quality of code and an object oriented approach for development. This types folder is used to store the types related to the parent component.

Note: global types (i.e types used by all components are stored in the common directory)

GroupTable.tsx:

Columns

Filters

Density

Export

Key	Default - English	NLD - Netherlands	DEU - German	GBR - English	IT - Italian
HelloWorld	Hello World	Hallo Wereld	Hallo Welt	Hello World	Ciao mondo
GoodMorning	Good Morning	Goedemorgen	Guten Morgen	Good Morning again	Buongiorno
good_evening	Good Evening	Goede avond			

Fig 5.6: group table

This is a very important component it receives 2 props :

- groupId
- languageList

This component uses the Datagrid library to showcase the data into a grid and it has additional custom functionality written by me, for editing different cells. For example editing a key, comment or default_text calls the graphql mutation to update the key. Whereas editing a language calls the mutation to update translation. And in case a key is added to an empty row a CreateKey mutation is called. This way a cell can have different function based on its column. The flow chart for this component can be seen in Diagram 5.1.

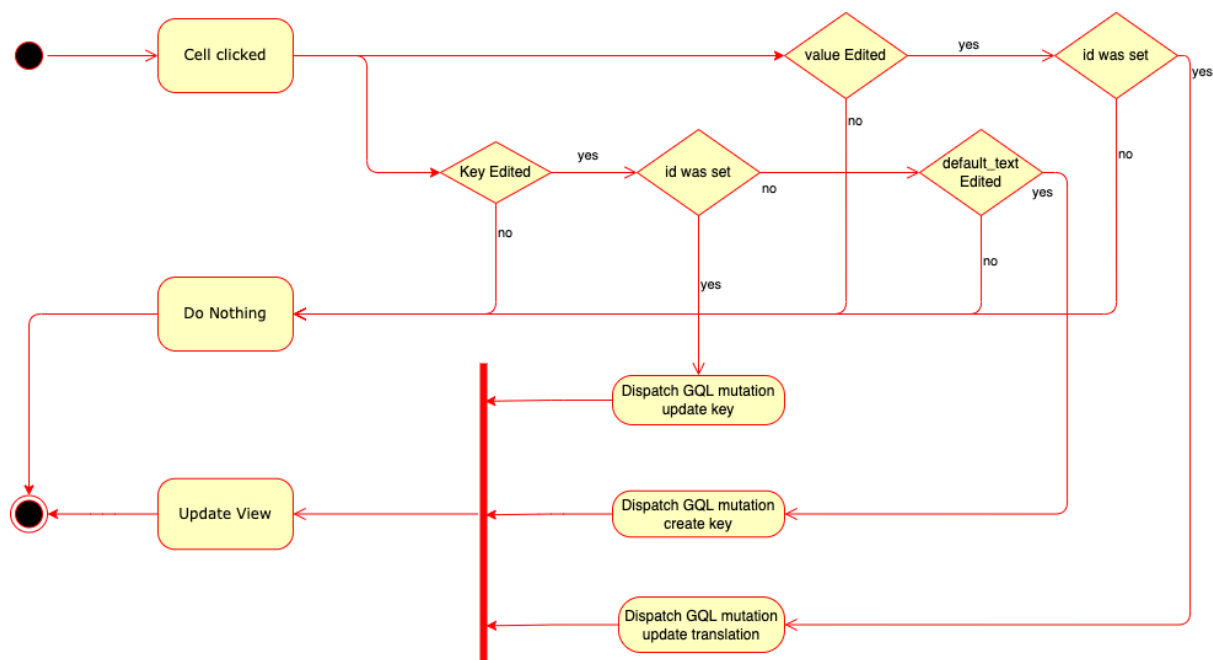


Diagram 5.2: Flowchart Cell edit functionality

For getting translations using the GoogleAPI there is custom functionality. As you can see in the figure there is a Google translate icon inside a cell. When the cell is in edit mode clicking this button triggers the request to Translate service and updates the cell.

I use the groupId prop to fetch the group data. When the selected group is changed on the front end it refetches the keys and translations.

Also the language list is provided as a prop because projects and groupTemplates can have different supported languages. So based on if this table components being rendered in Templates page or Project page the list of languages is provided by those pages.

Another important aspect of the code structure is how the main components view gets updated after a child component makes a change. For this purpose I create an update function in the parent component which queries the data. And pass the function down to a child component as a prop. To understand this better let us take the example of a group inside a project. So when we add or remove a language support inside a project the desired outcome is that the new languages be displayed on the group table. After the language list gets updated through a mutation, a positive response is received from the backend. The updateProject function is called this in turn queries the project again and updates the projects view.

Auth:

This module is responsible to authorise users when logging in or trying to access some settings which have role based access. An auth api is used to sign JSON web-tokens while authenticating a user. The web-token is later used to authorise requests to the backend.

Diagram 5.2 shows how a user gets the web-token while logging in with email and password.

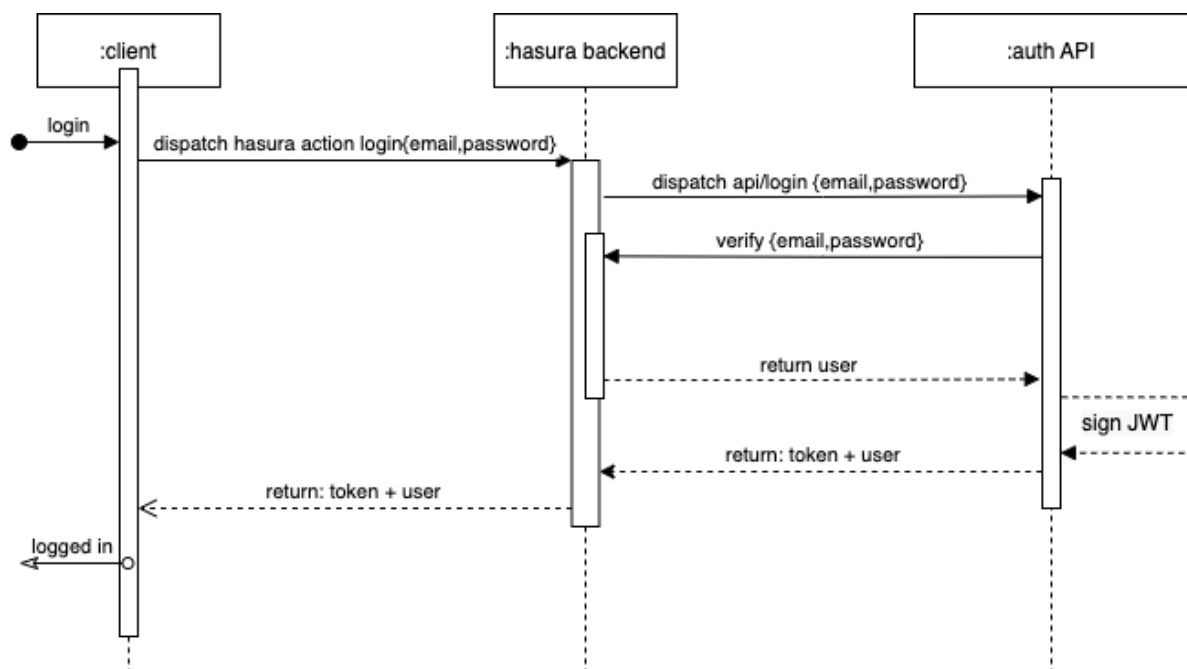


Diagram 5.3: Sequence diagram Authorisation API

5.5 Services:

The Integration library and Translation service were part of realisation, they have already been covered in the research section as these parts of the project were also part of research. Here are some charts to show their work flow.

Translation service:

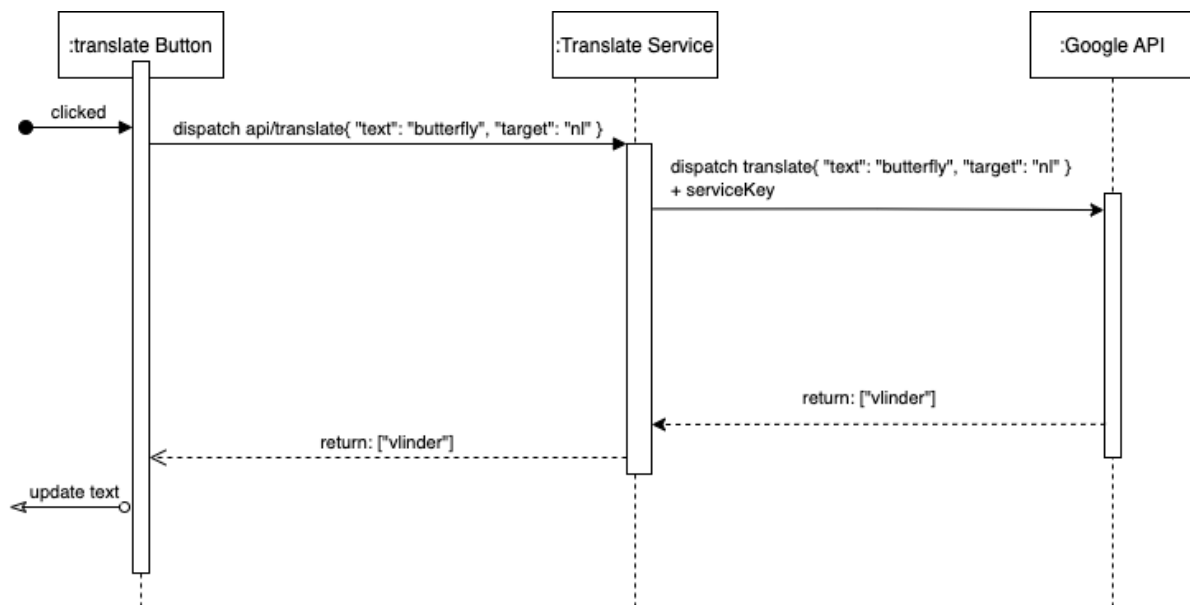


Diagram 5.4: Sequence diagram translation service

Integration library:

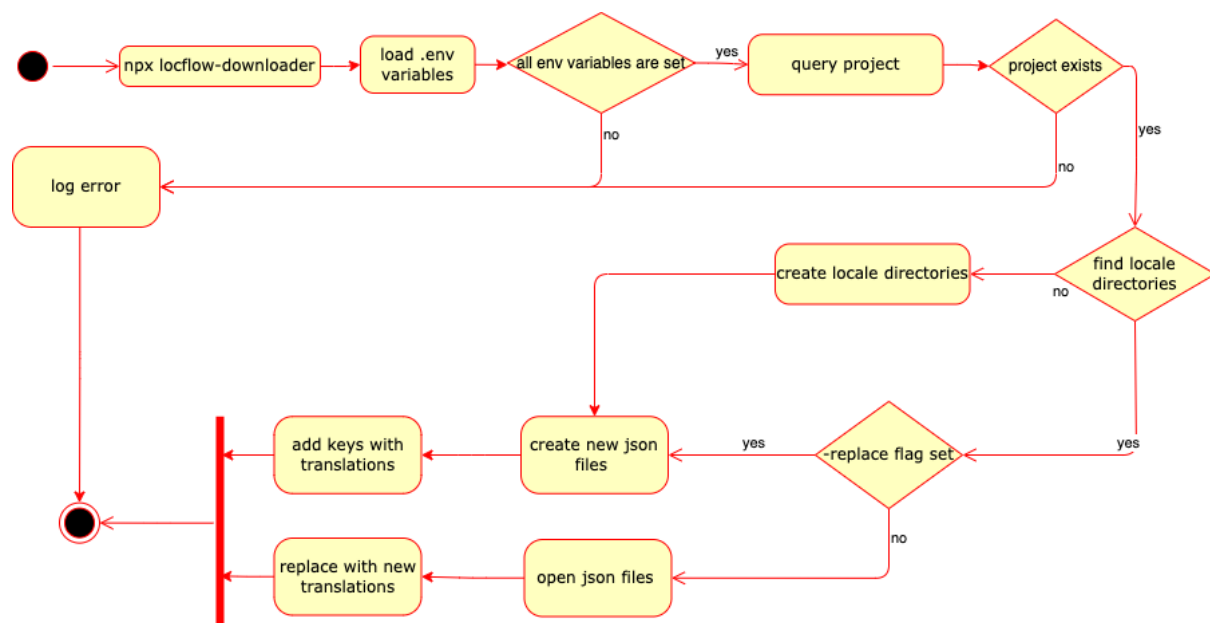


Diagram 5.5: Flowchart Integration library

5.7 Steps during realisation:

1. Set up the local project database and backend
2. Run sql code to setup database
3. Setup the hasura properties, connect to database and generate a graphQL schema
4. Setup frontend libraries, create styling files for the theme
5. Create all the pages and routes
6. Connect the pages to the sidebar
7. For every page start creating components
8. For each components functionality write the graphQL queries/mutations
9. Connect components to each other
10. Debug components
11. Create more components as needed
12. Debugging errors
13. Styling the code
14. Create the translate service
15. Create Google console account and set up the security keys with our service
16. Connect the translate service to the frontend component
17. Create the integration library
18. Create NPM package out of the library
19. Create a sample dev project to test the library
20. Install the library
21. Create a project on the platform with groups, keys and translations.
22. Test the library by integrating the platform keys & translations into our sample project.

This is a short overview of the steps carried in this order. In conclusions most of these steps required: research, learning, development, debugging, testing, rewriting, restructuring, guidance and documentation.

6. Testing

During development different components need testing strategies. This chapter focuses on documenting the test strategies used. The tests were carried out manually after finishing their relative story and the UI testing steps were done once every time before sending code to production. These test strategies also act as a definition of done to the requirements in chapter 3.2.4. The better way to do testing is to use an automated test also for UI. But as it was not a must requirement, the automation of UI testing was not focused. It will be advised in the advise section. The code and build however were tested automatically before creating a build using the gitlab pipelines and also locally before pushing to gitlab using linters for React, NextJS and Typescript.

6.1 Database and Backend:

To test the database and backend we need to prove that when a request is made to the backend the requested change is made in the database. And when something is queried from the backend the correct data is received from the database.

To do this testing I use the Hasura API explorer which allows us to write a graphql query/mutation and run it from there using a button.

- ✓ Test to query projects with their groups and keys.
- ✓ Test to add a translation for a key and check the added translation in Postico

6.2 Front-End UI testing:

To test this we need to manually do Quality Assurance tests on the working of different UI components and their edge cases.

- ✓ Dashboard shows correct number of projects and their correct relative statistics on load.
- ✓ Projects page shows all the projects enlisted and each project can be clicked to open the project
- ✓ Autocomplete on Project page shows the project name filled and does autocomplete the input.
- ✓ Create Project page shows the available templates, and languages to be selected.
- ✓ After creating a project the project can be seen enlisted in Projects.
- ✓ On templates page the autocomplete to find templates shows the templates on input.
- ✓ On templates page the create template functionality work and creates a new template.
- ✓ When deleting a template the templates get refreshed and the deleted template does not exist anymore
- ✓ Adding deleting languages to templates.
- ✓ On Languages page the supported languages by the platform are shown
- ✓ A new language can be added or a previous language can be deleted. The view refreshes on both of those actions.

- ✓ On the table a new translation can be added for a key
- ✓ A translation can be generated using the translate button
- ✓ A translation can be deleted
- ✓ When a language is added or removed the columns change accordingly
- ✓ A key can be added or removed
- ✓ After adding a new key a new empty field comes up
- ✓ Sorting filtering and hiding columns works correct
- ✓ The export JSON produces the JSON file as expected
- ✓ The View of the group refreshes after any changes
- ✓ On a specific Project page a new group can be added to this project
- ✓ A group can be bookmarked to save for quick access
- ✓ Bookmark removed correctly
- ✓ When deleting a project or group a popup shows up
- ✓ On agreeing to a delete popup the deleted group or project does not exist on the platform
- ✓ When the settings icon is clicked the settings options can be seen

6.3 Translate Service testing:

Testing the translate service can be done by making an api call to the translate service providing the correct query parameters. I use postman to test the query.

- ✓ The query returns the correct translation.

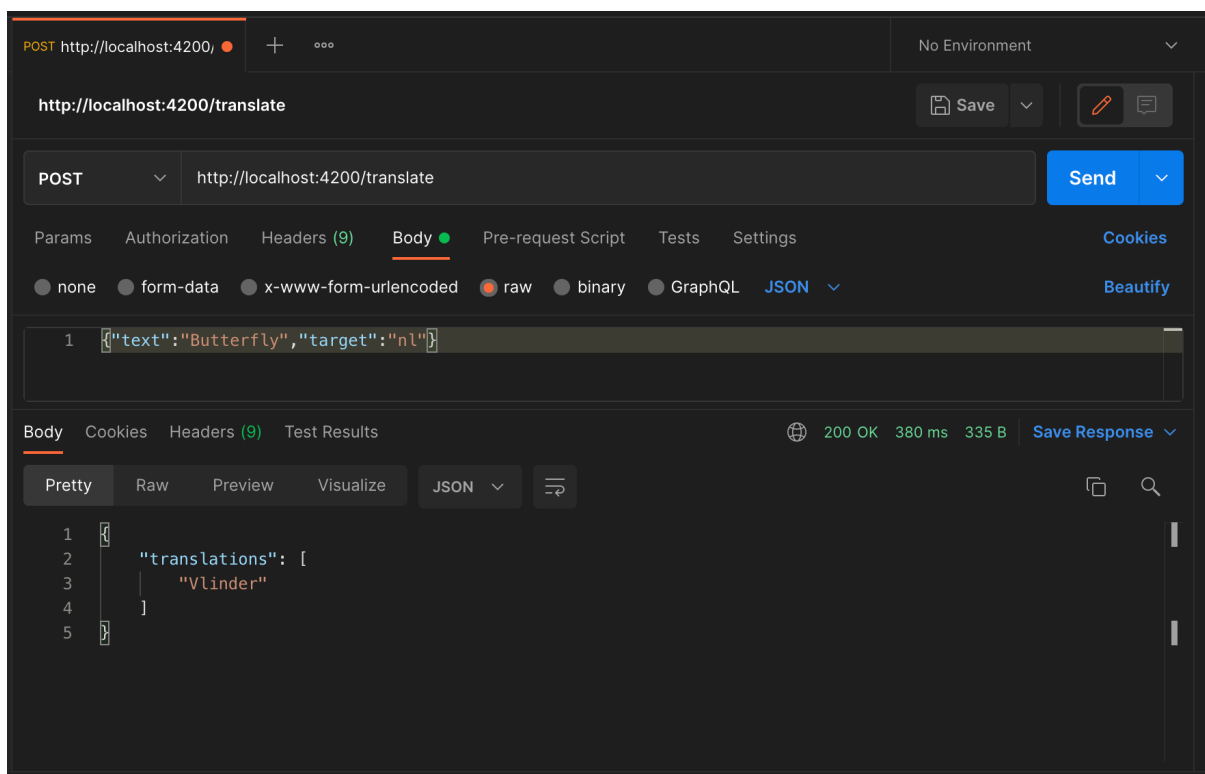


Fig 6.1: Postman API request

6.4 NPM Integration testing:

To test the integration a sample project was created. In this project the keys are used and the translation values for the keys are updated through the platform, after integrating the library it is checked if the updated version of translations can be seen on the sample project:

- ✓ NPM install adds the npm library to node_modules directory and package.json dependencies
- ✓ “npx locflow-downloader” adds the files to locales
- ✓ “npx locflow-downloader -replace” replaces the existing files

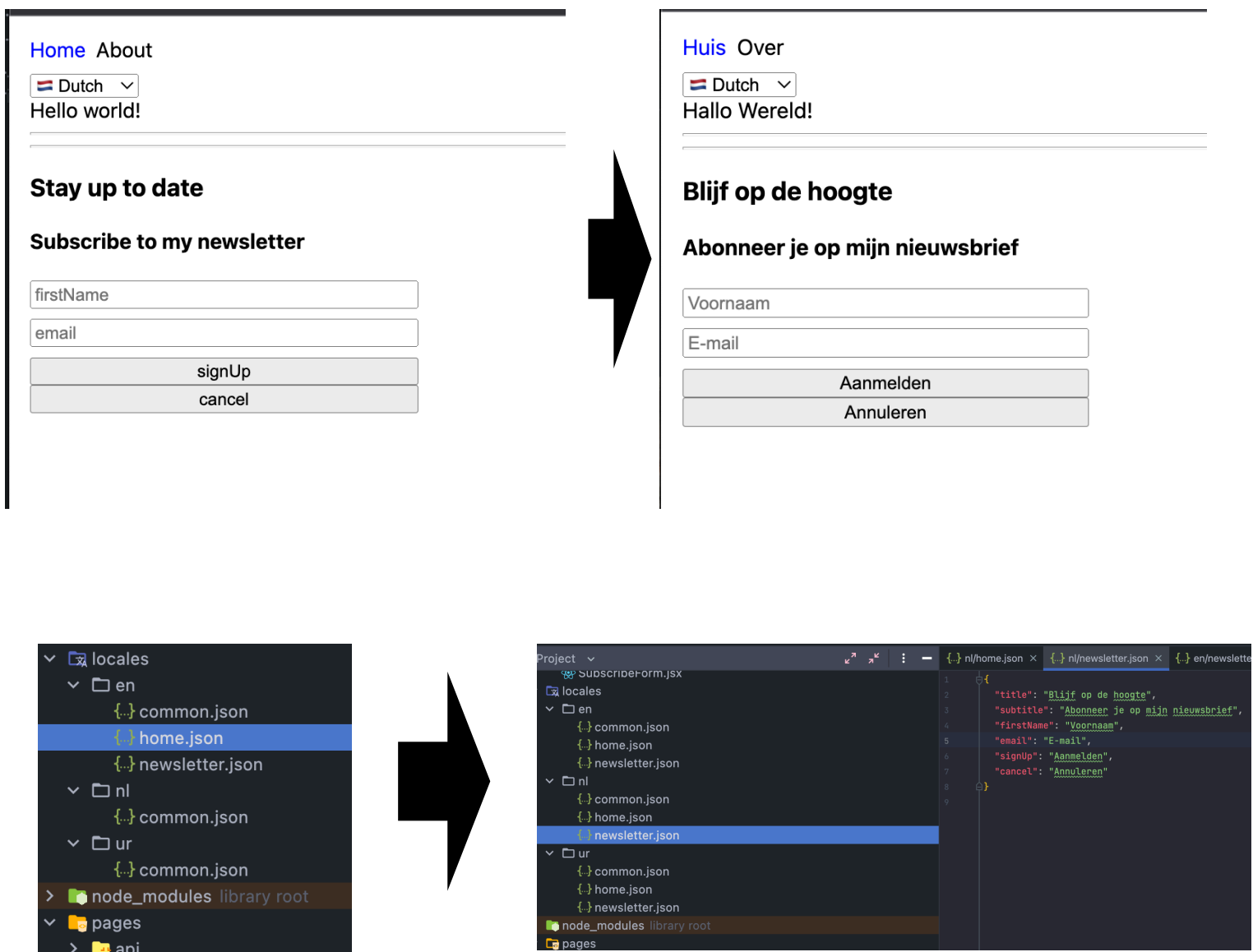


Fig 6.2: Screenshots of the sample project

7. Advise

7.1 Development Process

This advise is towards the developers for using the built platform. It aims on advising a workflow to make best use of the developed technology.

First of all we need to look at the current development process for localisation. So as it stands during development, the developers perform live translations. And check the project to see if the translation is right in the project. There is currently no reported systematic approach.

The developers look for hardcoded strings in the user interface that can be made dynamic to be translated individually. Consider a name for the translation key. And add manually the key and its translations into the relevant localisation json files for each supported language. After that, they go over the rendered UI to check if the translation comes up as expected.

Now as part of this project it would be advised to adhere to a standard procedure for the localisation process. Firstly to keep track of different translations to be done. At the planning stage stories in the Jira board should be created for translation tasks. At the moment in a project every different module or page is separated as a good practice, from project structure point of view. It is advised that every separate module or page has a separate Group in the Localisation platform. One story per Group should be a standard to keep the stories short and precise.

As a platform feature there is support for templates. If there are recurring keys and groups, they can be saved into a template. During the planning, a story to create a template should be created. These templates can then be reused inside a project with the keys and their relative translations copied to the project.

When the planning phase is done, next comes the development phase. During the development phase the first step should be setting up the localisation library in the project. And to test the library one simple manually written file with one or two keys and at least 2 languages supported should be added. Now that we are sure the library is working fine in the project. We can create a new project on the localisation platform that was created during this project.

While creating a project on the platform, we can select some templates that are commonly needed for most projects. For example a general template that has keys like Login, Log Out, Sign Up, Agree, Decline etc. After creating the project on the platform we can start creating the needed Groups and their keys. For each key, we can use auto translation feature to get translations for different languages from the Google API.

Now that we have the project and the required Group setup on the platform we can use the NPM library for integration of translations into our development project. This library was custom created during development of this project. To setup the library we run NPM install

locflow-downloader (the library name). When it is installed, we setup the required environment variables in the .env file. In the .env file it is required to provide the project id from the platform, the GraphQL endpoint where the backend for the platform is running and the path to the locale folder in our development project. After this setup, by running the NPM command `npx locflow-downloader`, we get all the translations and groups automatically downloaded and written to our locale folder. After this command we will have all keys translated in their relative language directories in our project's locale directory.

7.2 Testing Process

Following the development stage, the story can be assigned to the quality assurance. The QA then tests, each supported language for the module or page which the story specifies. If QA testing goes as expected, the QA can finally approve the story for deployment. It is advised to the developers to always use camel case while creating the key. This helps the QA in finding any bugs on the UI, because when a translation is not working properly the string on the UI shows the key instead of the translated value. And it is easy for QA to see any camel case string on the UI while testing

To test the user interface a UI-test-automation tool could be incorporated into the project. For example Selenium is an open source umbrella project providing a collection of browser automation tools and libraries. It provides a playback tool for building functional tests across most modern web browsers without the need to learn a test scripting language.

7.3 Scaling

When the organisation grows, it is advised to pass the translation tasks to professional people completely responsible for translations. These people can use the platform with an account with a translator role which has less privileges than an account with admin role.

As the company has a desire to upscale this project in future as a SaaS platform. It is advised to add organisations to the database with different feature access based on their plans. The early research on existing platforms completed during this project can also be utilised as a guide for pricing the plans. The detailed research on existing platform can be found in the Appendix.

7.4 Continuation

This is already in the plans of organisation to possibly scale the project to a SaaS platform. Even though it was not a requirement for the current project, most of the architectural and business design was made keeping that continuation goal in the mind. So my advice is to first use the platform internally, to gain experience on its suitability and discovering potential shortcomings. If it has proven itself internally, then it might be time to build upon the main features to scale it to a SaaS platform.

As we know the translation service is based on usage of Google API which has a usage based cost plan. If the project is continued as Software service, my advice would be changing the translate service in such a way that the user of the service uses their own Google API

credentials. This would help keeping the cost down, as the Google API only charges for higher usage. If each user has their own account with credentials the usage cost would be low per user.

Another feature idea from me was to integrate a VS code plugin for synchronising translations. Which I believe was a nice idea but was out of the scope of the project. The recommendation is to create a fun hackathon challenge out of this. Or if it becomes a highly desired feature then plan a small project to develop the VS code plugin.

The NPM integration package can be expanded and made into a bigger library with features like, uploading the local project, live changes or addition of automated testing of keys and translations.

8. Conclusion

This graduation project, after a period of learning to solve new problems, and motivating challenges, comes to an end. In this chapter, I would summarise the entire project in order to reach a conclusion.

I had no idea about many aspects of the project at the start, and it seemed a bit overwhelming. The situation with the localisation process at the time was inefficient. The translations had to be typed into the JSON files by hand. Someone had to create the translations for each language as well. It was not fun to do that manually or with a quick translation app. Developers disliked this part of the process because their primary goal was to spend more time writing code. Apart from the difficulties in the development procedure, management was also a major issue, because all of the company's client projects exist in their own repositories, so the translation files are very scattered.

After being given the opportunity to work on this tricky project, I examined all of the tools and skills I had acquired during my Bachelor's degree in Software Engineering. First and foremost, we needed to create a good project plan so that we could have a guideline throughout the process of how progress is being made and which steps should be taken.

The next mission was to comprehend the problem as thoroughly as possible. I formulated some research questions based on my limited knowledge from the assignment description to accomplish this. The research subquestions were answered methodically in the research section. However, I would like to address the main research question in this conclusion.

Main Question: How can we centralise the management of translations for many projects?

Well, based on the advice report and study on subquestions, we can conclude that a platform to manage translations in one place was indeed required. The solution to how it can be done was a continuous learning process throughout the project's development. And now we have an answer in the form of this project.

At the graduation level, many tasks from a Software Engineer Professional's daily work were completed. First, I produced an advise about existing frameworks. To generate a requirement analysis, I employed several analysis methodologies such as surveys, interviews, library research, and field research. A full design of the product was produced using my knowledge from my studies, taking into account architecture, business, schema, and user experience. The largest part was the development phase, which included tasks in which I was proficient as well as some that I had never done before, which was a valuable learning experience. Finally, I offered another piece of advice on the development process and the project's future.

Finally, I must say that this was a challenging project. Knowing how significant it is to the company, how it will solve a big problem, and how everyone was looking forward to it makes me happy. And it gives me the confidence to create valuable software solutions in my profession.

References

- [1] i18n- <https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/i18n>
 - [2] i18next docs - <https://www.i18next.com/>
 - [3] Research Methods: What are research methods? <https://libguides.newcastle.edu.au/researchmethods>
 - [4] HBO-i - The Dot Framework https://ictresearchmethods.nl/The_DOT_Framework
 - [5] scrum.org - What is scrum - https://www.scrum.org/resources/what-is-scrum?gclid=CjwKCAjws8yUBhA1EiwAi_tpEQFMvx3IYO2kEivTotChPyAXL2eml03rC1GiDtJbBxkYDIEu65rh-hoC2BEQA_vD_BwE
 - [6] ProfitFlow - <https://profitflow.nl/>
 - [7] Team Asana. (2021). What is a risk register: a project manager's guide <https://asana.com/resources/risk-register>
 - [8] Localise - <https://lokalise.com/>
 - [9] Locize - <https://locize.com/?lng=en>
 - [10] POEditor - <https://poeditor.com/>
 - [11] Charles Lane & Nico Kruger. (2021). How to Write a Software Requirements Specification [https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document#:~:text=A%20software%20requirements%20specification%20\(SRS\)%20is%20a%20document%20that%20describes,\(business%2C%20users\)%20needs.](https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document#:~:text=A%20software%20requirements%20specification%20(SRS)%20is%20a%20document%20that%20describes,(business%2C%20users)%20needs.)
 - [12] Creating Node.js module - <https://docs.npmjs.com/creating-node-js-modules>
 - [13] Scott Robinson. Reading and Writing JSON Files with Node.js - <https://stackabuse.com/reading-and-writing-json-files-with-node-js/>
 - [14] Debashish Pal. (2019). Publish a npm package locally for testing - <https://medium.com/@debshish.pal/publish-a-npm-package-locally-for-testing-9a00015eb9fd>
 - [15] Janne Kemppainen. (2021). Make an NPM Package Executable with npx - <https://pakstech.com/blog/npx-script/>
 - [16] Google cloud docs. <https://cloud.google.com/translate/docs/basic/translating-text>
 - [17] Richard Koret. (2021). How to get started with the Google Translate API - <https://bdtechtalks.com/2021/07/04/google-translate-api-beginners-guide/>
 - [18] Google API Language support - <https://cloud.google.com/translate/docs/languages>
 - [19] MUI docs - <https://mui.com/material-ui/getting-started/installation/>
 - [20] Apex Charts - <https://apexcharts.com/docs/installation/>
 - [21] Hossein Shams. (2014). MVCC: An Architectural Pattern for Developing Context-aware Frameworks - <https://www.sciencedirect.com/science/article/pii/S1877050914008904>
-

Glossary

i18n - Internationalisation	HR - Human Resource
MKB - Small and medium-sized enterprises	CTO - Chief Technology Officer
ERP - Enterprise resource planning	Jira - an app that aids scrum process
B2B - Business to Business	QA - Quality Assurance
DOT - Development Oriented Triangulation	Localisation - making something local in character
ICT - Information and communications technology	i18next - npm library for internationalisation
MoSCoW - Must Have, Should Have, Could Have, Won't Have this time	iOS - Apple phone operating system
SCRUM - a framework for developing software	CLI - command line interface
Sprint - a short, time-boxed period when a scrum team works to complete a set amount of work	dev - developer
Story - description of features of a software system.	JWT - JSON web token
NPM - package manager for the JavaScript	CDN - content delivery network
API - application programming interface	UI - user interface
	MUI - Material UI
	MVCC - Multiversion concurrency control
	CRUD - create, read, update, and delete
	RBAC - role-based access control

Versioning

0.1	Start Graduation Thesis Document	Mon, 14 Mar 2022
0.2	Introduction	Tue, 15 Mar 2022
0.3	Organisation	Wed, 23 Mar 2022
1.0	Requirement Analysis	Fri, 25 Mar 2022
1.1	Research on existing solutions	Wed, 6 Apr 2022
1.2	Architecture + DB design	Wed, 20 Apr 2022
1.3	Research on Google API	Mon, 25 Apr 2022
2.0	Research Integration	Wed, 11 May 2022
2.1	Got feedback	Thu, 19 May 2022
3.0	Fixed Report	Fri, 27 May 2022
3.1	Report submission	Sun, 29 May 2022
4.0	Got feedback	Thu, 2 Jun 2022
4.1	Fixed Report for final version	Fri, 10 Jun 2022
4.2	Final Submission	Sun, 12 Jun 2022

Version Table

Appendix

Appendix I - Planning

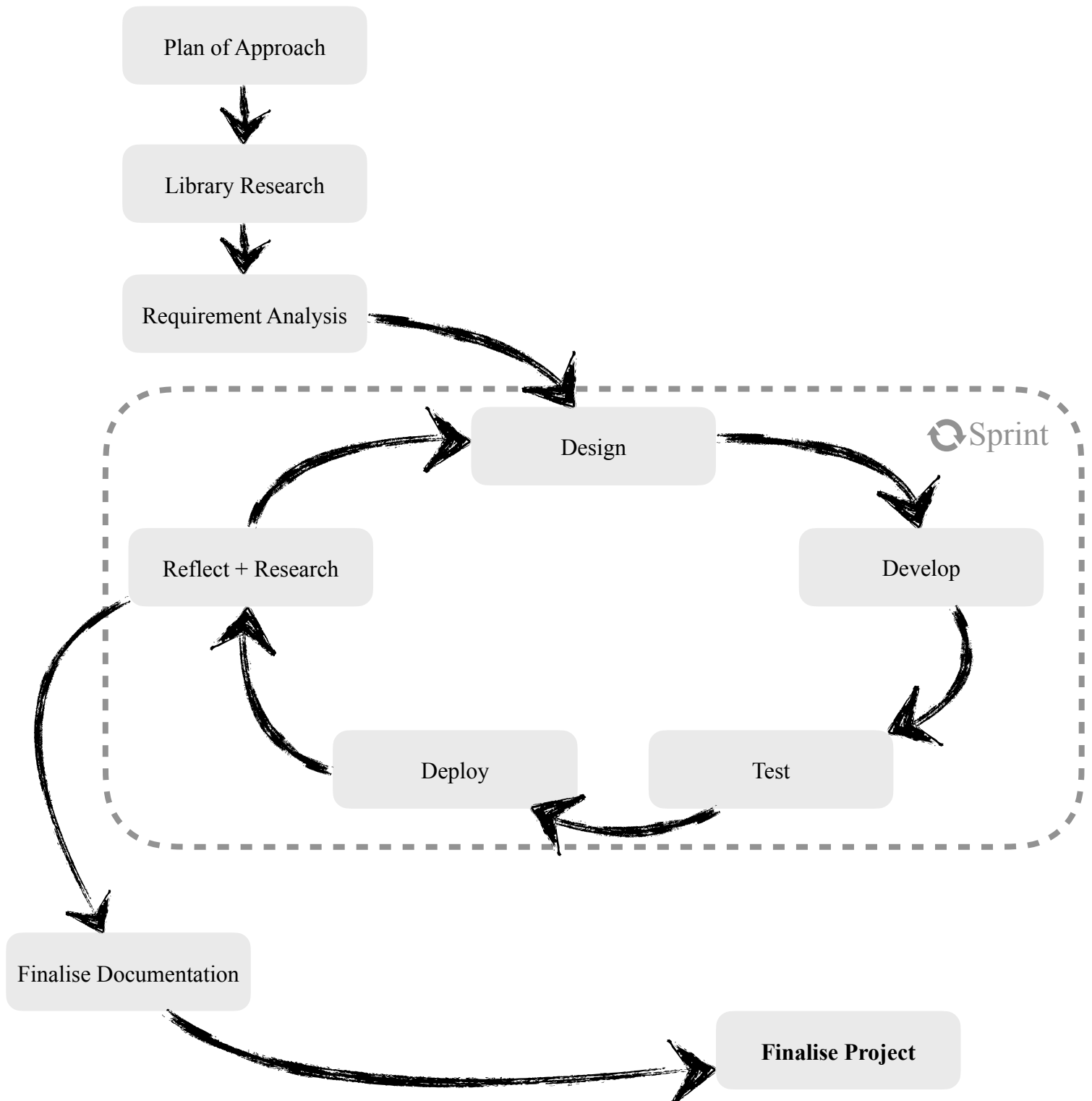
Appendix II - Survey

Appendix III - Advise Report

Appendix IV - Database design summary

Appendix I - Planning

Steps:



Timeline:

Week Nr	Date	Task	
1	Feb 7	Intro week	
2	Feb 14	Plan of Approach	
3	Feb 21	Plan of Approach	
4	Feb 28	Library Research + Requirement Analysis	
5	Mar 7	Jira + Sprint Planning	
-	Mar 13	Deadline Final Plan of Approach	
6	Mar 14	Sprint 1	
7	Mar 21	Sprint 1	
8	Mar 28	Sprint 2	
9	Apr 4	Sprint 2 + Research	
10	Apr 11	Sprint 3	
11	Apr 18	Sprint 3	
12	Apr 25	Sprint 4	
13	May 2	Sprint 4 + Research	
14	May 9	Sprint 5	
15	May 16	Sprint 5	
16	May 23	Finalise Concept Version	
-	May 29	Deadline Concept Graduation File	
17	May 30	Work on the Feedback	
18	Jun 6	Work on the Feedback	
-	Jun 12	Deadline Final Graduation File	
19	Jun 13	Product integration + Presentation Preparation	
20	Jun 20	Presentation Preparation	
21	Jun 27	Graduation Session	

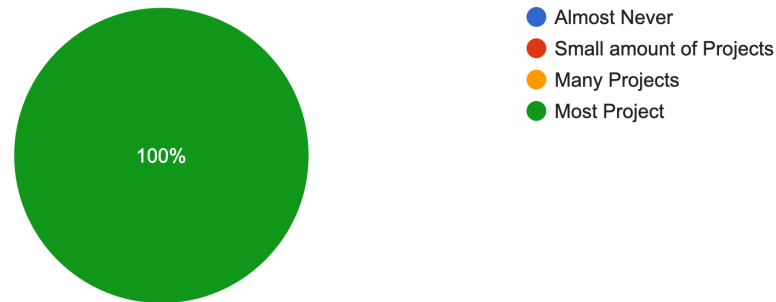
Table 1: Timeline

Appendix II - Survey

These are the results of a survey conducted on all the software developers working for ProfitFlow. The questions with choices show the relevant data diagram of the answers, meanwhile there were a few open questions for which the answers were written by participants, so rather than writing all the answers, they were summarised.

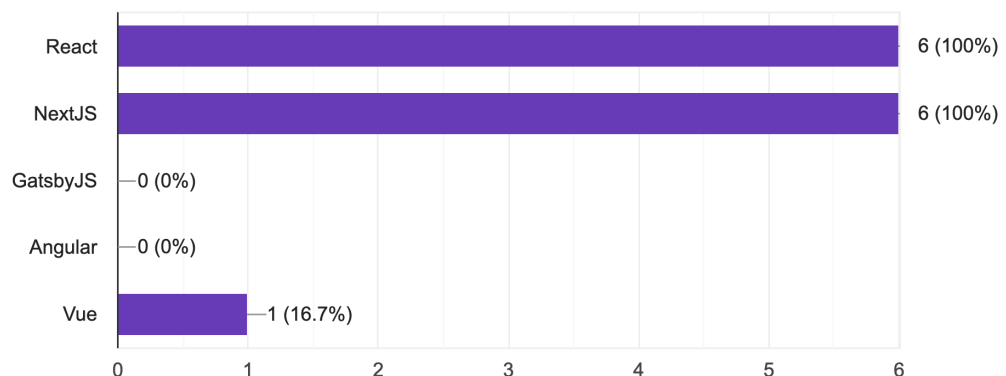
How often do you use Internationalisation in projects?

6 responses



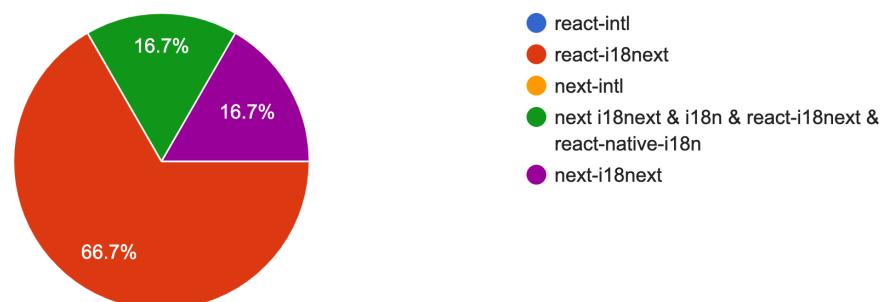
Which frameworks do you use for the front-end projects?

6 responses



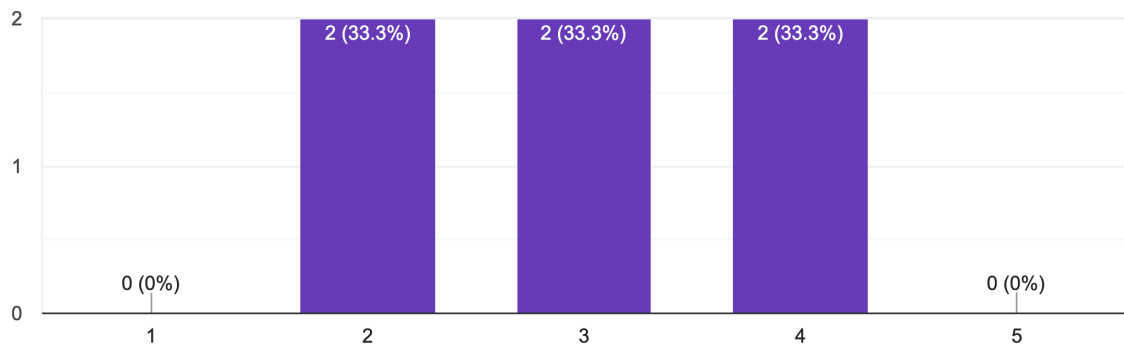
Which library do you use for Internationalisation in a Project?

6 responses



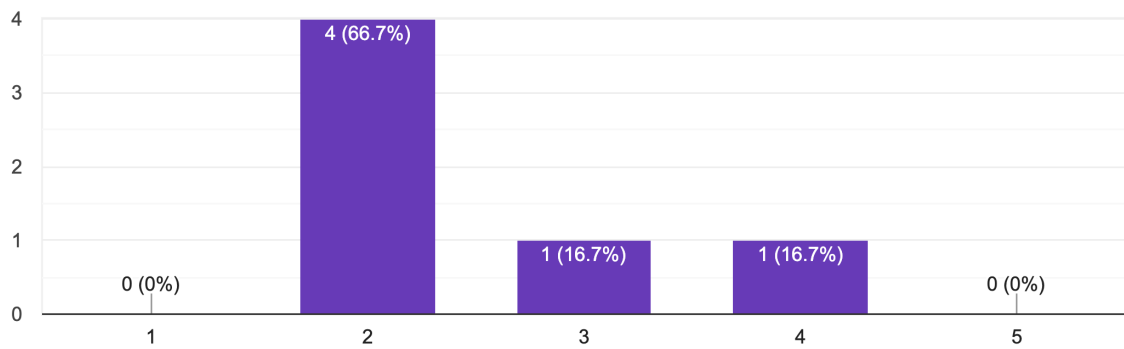
How convenient is it to program the translation keys into the front-end?

6 responses



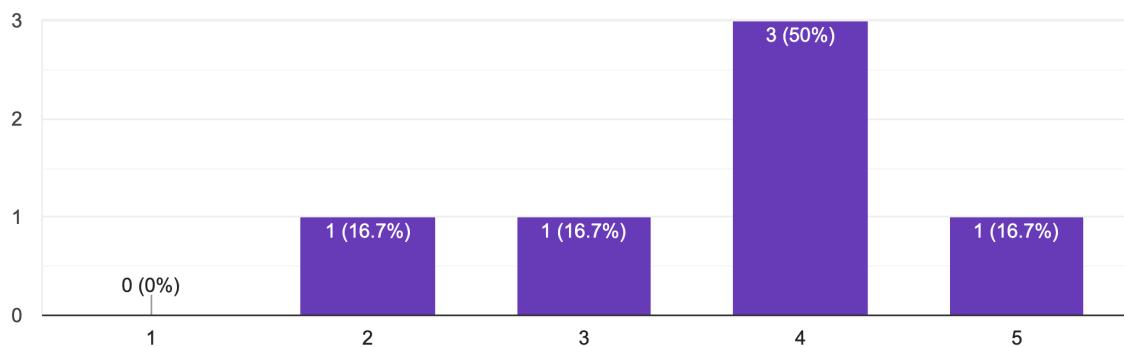
How difficult is it to update a translation or add support for a new language?

6 responses



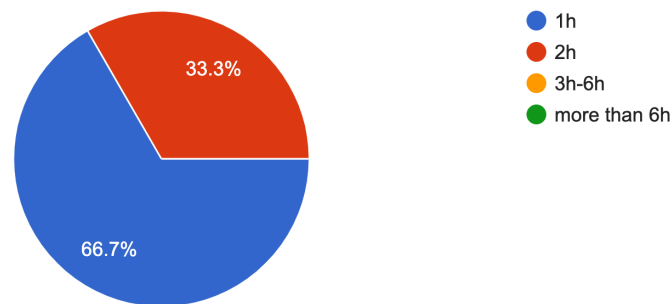
How difficult is it to test translations?

6 responses



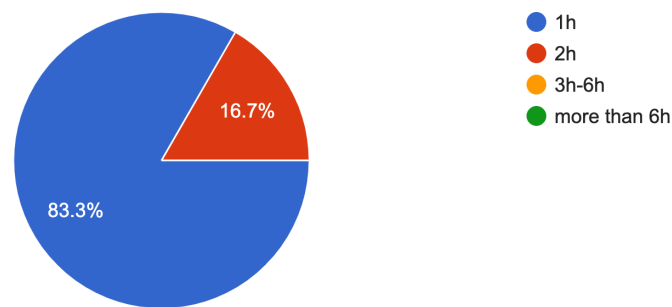
How much time per day do you spend on adding translations to a project for the first time?

6 responses



How much time per day do you spend on updating translations for a project?

6 responses



How do you currently add or update translations? (Process)

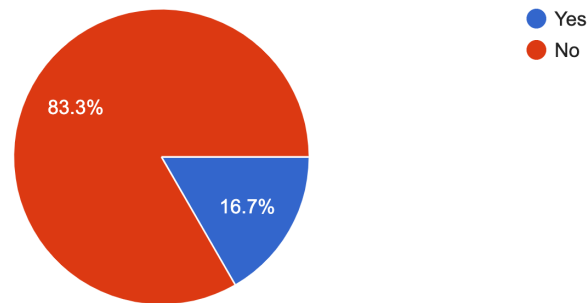
1. Look for hardcoded or ui strings which could be stored and translated separately.
2. Think of a name for the translation key (no real convention for translation keys at the moment)
3. Add the translation key and the translations to the necessary localisation json files (ex. Dutch and English).
4. Test if the translation comes up as correct in the project.

How is a project updated when a new language is to be made available in the project?

- Copy pasting the key/value "template" of an existing language and changing all the values manually.

Have you ever used a translation platform for the management of translation before?

6 responses



If you have used a platform for translations before which platform was it?

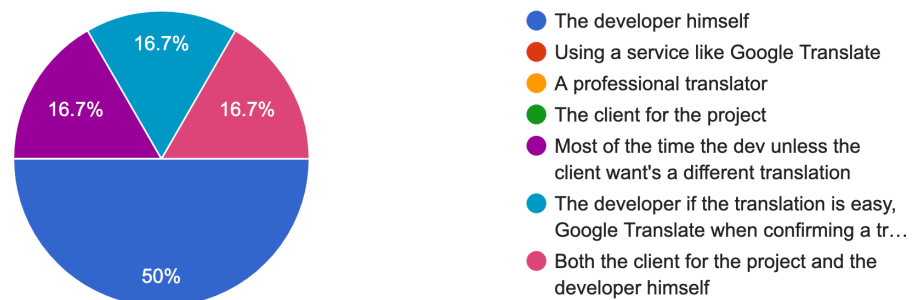
- Tonguetied: <https://tongue-tied.co.uk/>, which is a bit of an old application to create, update, delete and monitor translation keys.

What would be the ideal process from development point of view for translations of projects?

- Either via a third party program where you can click a flag of a language you'd like to add and then all the existing translation keys will be duplicated for a new language entry and have all their translation strings automatically translated, via tools such as Google Translate or AI. If not using a third party program, an automatic translation of the existing locale, such as autocomplete when filling in the new language' respective translation strings.

Who decides the translations for different languages?

6 responses



Which features can you suggest for a platform used for management of translation?

1. A way to test translations. In the project itself but maybe also on the platform.
2. There should not be any duplicate keys on the platform.
3. Have a common convention for translation keys(maybe make a translation key generator?)
4. Search and filter functionalities.
5. Maybe include comments on the platform for translations.
6. Track who made translations(Some sort of history on translations would be ideal)
7. A CI/CD process for all projects which updates translation keys of the project automatically if there are updates.
8. Should automatically check if any translation for a language is missing.

Appendix III - Advise Report

Advise Report

LocFlow - Localisation

Rehan Ahmed

Supervisor - **Joey Teunissen (Technical Director)**

ProfitFlow.

Maagdenburgstraat 14

7421 ZC Deventer

+31640587446

Version 3

Table of contents

1. Introduction	62
1.1 Report structure	62
2. Analysis of available products	63
2.1 Research design	63
2.2 Available products	63
2.3 Conclusion	65
3. Advice on development process	67
3.1 Research design	67
3.2 Development Process	67
3.2 Testing Process	68
3.3 Scaling	68
Appendix IV - Database design summary	69

1. Introduction

This is an Advice Report for the Graduation Project in relation with the graduation thesis. This graduation is for Bachelors of Software Engineering with specialisation in Big Data. The internship for the Graduation project is performed at ProfitFlow. B.V Deventer.

The majority of the websites built for customers use NextJS as the front-end, Hasura as the back-end, and PostgreSQL as the database. However, depending on the needs of the consumer, alternative front-end, back-end, and database technologies may be employed.

Many customers want websites and apps to be available in a variety of languages. Internationalisation, also known as i18n, is the process of making a technology product available in multiple languages. ProfitFlow's i18n is currently done in a less efficient manner. The i18n files for each project are saved locally within that project's repository.

This is where the research and development project for this Graduation comes in. The goal is to investigate the present methods for i18n development and utilisation, as well as to improve the i18n development process. Furthermore creating a platform that serves as a translation directory. This can be useful both during development and when using i18n in projects.

1.1 Report structure

In this Advice report there is advice on different aspects at different phases of the whole Graduation project. Following is an overview on the contents of this advice report:

- Analysis of available product:

This is an Analysis of existing solutions available for the same problem. It covers the costs aspects and the features available with the existing platforms. This part of the advice is delivered at the start of the project before the requirement analysis is conducted to get an understanding of the existing solutions and reasoning on why a research and development project about a translation platform is needed.

- Advice on software development process, including the test process.

This is an advice on how to best use the platform after its development. This part is Advised after the development of the platform. And its aim is to advise a better workflow for the translation process.

- Advice on the test process.

This part of the report is a guide for the testing work flow during a development process

- Advice on scaling.

This part of the advice is concerned at mentioning some possible improvements during scaling of the project

2. Analysis of available products

2.1 Research design

Focused Research question:

- What are the available existing solutions and how do they compare to each other?

Research Methods:

Library research

- Literature study
- Available product analysis
- Best good and bad practices

2.2 Available products

Lokalise

Lokalise is a multinational firm established in Riga, Latvia. It's a software platform for managing technical product translations and localisation. It's a platform that runs on the cloud. The software makes it easier to translate mobile apps, games, and websites. Its primary goal is to improve the technical project development process.

To use Lokalise in a react project, first create an account on Lokalise web-app. Lokalise provides a CLI that can be used to create projects, upload, and download translation files. From the Lokalise web-app we can get an API token which is used for the authentication of requests from the CLI. Now after setting up the Lokalise CLI using the API token inside our project folder, we can make request to upload or download the translations stored on the platform. The files are downloaded in JSON format which can be used easily with any of the react i18n library.

As Lokalise is a software as a service platform there are price aspects which need to be taken into account for this advise report. There are different tiers of monthly payment plans to choose. First one is called Start it costs \$120 per month there is no limit of projects but there is a limit of 5000 hosted keys it includes integration support with different developer softwares like Git, Slack Web-hooks etc, furthermore there are 10 seats included which means 10 people can collaborate on one plan. Next plan is called Essential it costs \$230 per month, just like Start it also includes 10 seats, the limit for hosted keys on this plan is 10000. Apart from this there are more features and integrations. Features like translation history, Machine translations, Stats and reports. The integrations include Jira and Wordpress. The next plan is called Pro which costs \$585 per month. This plan includes 15 seats, 30000 hosted keys. Features like Branching and User Groups. Asana, Azure, Amazon S3 and Google Cloud Storage integrations. Last tier of the plan is called Enterprise and to get the price for this one must contact their sales team for a custom quote.

Locize

Locize is an online web-based Translation management system. Owned by inweso GmbH based in Switzerland found in 2012. It is used for management of translations for tech products, helping the development process. The creators of locize are also the developers of i18next library. It is a Javascript based SaaS web-app.

There are many features provided by Locize. There are statistics on the modifications done, translations retrieved or words per language. There is history of changes and who did the change. Where we can do translations ourselves we can also order translations from Locize. There is availability for Machine Translations. And there are some more useful features.

There are several ways to use the translations from the platform. First is the simplest way to manually download the translations in JSON format and place them in our project folder. The other way is by automation, Locize provides an API which can be used to sync with our local code and then there is also a Sync tool. Locize platform is fully compatible with i18next library.

As this is a platform which provides service for a cost there is also a pricing structure. There is a basic \$5 starting price and then the users are charged based on their usage. There are different metrics used for calculating the usage based price. The users get charged monthly per word, and then charged per modification and per download. First 10,000 words are charged \$0.004 per word and then per every next 20,000 words it keeps getting low by \$0.001. First 1,000 modifications of the month are charged \$0,02 per modification and then for every next 2000 it decreases by \$0.005. The first 500,000 downloads are charged \$0.0001 per download, next million download is charged \$0.000075, after that every download is charged \$0.00005. After combining all these costs the monthly costs are calculated.

POEditor

POEditor is another competitor in the localisation platform market. It has many big brands as their users. It is written in the C++ language. And also facilitates in handling translations for different software products. It is very famous in the WordPress development community. But can also support IOS, Android, Angular, React and many more in their translation process.

POEditor just like the previous two comes with its own features. Smart translation memory reduces the translation work volume. It has integrations for Github, GitLab, BitBucket, Slack, MS Teams. And the common features like Automatic translation and translation history.

For the development process when using POEditor it supports OpenAPI to make API calls for different Actions. It has API calls to sync the project or to Export translation files from the platform to our project directory.

POEditor has a simpler pricing model. There are different tiers of payment plans, but the only thing considered is how many strings are stored on the platform there is no limit on API calls, project, contributors or languages. For 1000 strings there is no fees. For up to 3000 strings is the starter plan which costs \$15 per month. And there are plans with string limit of 10,000, 30,000, 100,000. Priced at \$45, \$120, \$200 respectively.

2.3 Conclusion

As these previously mentioned platforms are hosted by well known companies availability, performance, and security are well taken care of. When it comes to scalability it should also not be an issue to scale with these platforms to a reasonable extent as all of these platform have support for many projects at their high tier plan.

But scalability comes at a cost, although it can be seen that it becomes cheaper per word with every level of upgrade to the payment plan. The issue here is ProfitFlow as a company needs to scale very fast as there are regularly new clients with new projects, which are long-term managed by the company itself. So as the number of clients and the projects starts increasing it can reach to a level where managing all projects under one payment plan is not reasonable from the cost and the management perspective.

As there can be a situation where a client wants to change the company that manages the project, in that case the management of translations for that project would also need to be handed over with the product. And removing the project from the platform would cost the company. Because the income from the client for the translation would stop but ProfitFlow would still be paying the fee for the monthly plan for the platform. Apart from that moving the project from one account to another is not very easy to manage. Where having a different account on the platform for every customer makes more sense but that increases the costs as for each customer there needs to be a separate plan.

Another advice is about the way the keys are stored. As a feature there could be possibility to have some common translation keys for free e.g sign-in. So the common translation keys could be used by any user without paying for them. This can be a competitive advantage and help in management of common keys.

	Integration	Scalability	Pricing Model	Pricing
Lokalise	Different mobile apps, web-apps, games	Limited number of allowed developers	Monthly fixed price for limited number of words & limited number of developers	\$585 30,000 words 15 developers
Locize	i18next library	Costs increase with higher usage	Price per word, per modification & per download	\$0.004/word for 10,000 words + \$0.02/modification for 1000 + \$0.000075/download if downloads > 1 million
POEditor	IOS, Android, Angular, React and many more	Limited number of words	Monthly fixed price for limited number of words	\$200 100,000 words

Table 1: Comparison of existing platforms

3. Advice on development process

3.1 Research design

Focused Research question's:

- How can our solution be integrated into existing and new projects?
- How is it possible to integrate Google Translate into our own platform?

Research Methods:

Library research

- Literature study
- Best good and bad practices

Field research

- ~~Explore user requirements~~
- Task Analysis

Lab research

- Usability testing

3.2 Development Process

This advise is towards the developers for using the built platform. It aims on advising a workflow to make best use of the developed technology.

First of all we need to look at the current development process for localisation. So as it stands during development, the developers perform live translations. And check the project to see if the translation is right in the project. There is currently no reported systematic approach.

The developers look for hardcoded strings in the user interface that can be made dynamic to be translated individually. Consider a name for the translation key. And add manually the key and its translations into the relevant localisation json files for each supported language. After that, they go over the rendered UI to check if the translation comes up as expected.

Now as part of this project it would be advised to adhere to a standard procedure for the localisation process. Firstly to keep track of different translations to be done. At the planning stage stories in the Jira board should be created for translation tasks. At the moment in a project every different module or page is separated as a good practice, from project structure point of view. It is advised that every separate module or page has a separate Group in the Localisation platform. One story per Group should be a standard to keep the stories short and precise.

When the planning phase is done, next comes the development phase. During the development phase the first step should be setting up the localisation library in the project. And to test the library one simple manually written file with one or two keys and at least 2

languages supported should be added. Now that we are sure the library is working fine in the project. We can create a new project on the localisation platform that was created during this project.

While creating a project on the platform, we can select some templates that are commonly needed for most projects. For example a general template that has keys like Login, Log Out, Sign Up, Agree, Decline etc. After creating the project on the platform we can start creating the needed Groups and their keys. For each key, we can use auto translation feature to get translations for different languages from the Google API.

Now that we have the project and the required Group setup on the platform we can use the NPM library for integration of translations into our development project. This library was custom created during development of this project. To setup the library we run NPM install locflow-downloader (the library name). When it is installed, we setup the required environment variables in the .env file. In the .env file it is required to provide the project id from the platform, the GraphQL endpoint where the backend for the platform is running and the path to the locale folder in our development project. After this setup, by running the NPM command npx locflow-downloader, we get all the translations and groups automatically downloaded and written to our locale folder. After this command we will have all keys translated in their relative language directories in our project's locale directory.

3.2 Testing Process

Following the development stage, the story can be assigned to the quality assurance. The QA then tests, each supported language for the module or page which the story specifies. If QA testing goes as expected, the QA can finally approve the story for deployment. It is advised to the developers to always use camel case while creating the key. This helps the QA in finding any bugs on the UI, because when a translation is not working properly the string on the UI shows the key instead of the translated value. And it is easy for QA to see any camel case string on the UI while testing

3.3 Scaling

When the organisation grows, it is advised to pass the translation tasks to professional people completely responsible for translations. These people can use the platform with an account with a translator role which has less privileges than an account with admin role.

As the company has a desire to upscale this project in future as a SaaS platform. It is advised to add organisations to the database with different feature access based on their plans. The early research on existing platforms completed during this project can also be utilised as a guide for pricing the plans.

Appendix IV - Database design summary

This is the first version of db schema design:

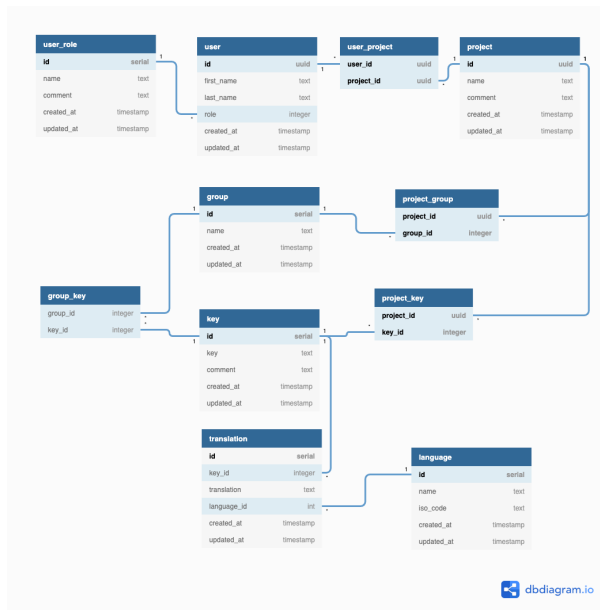


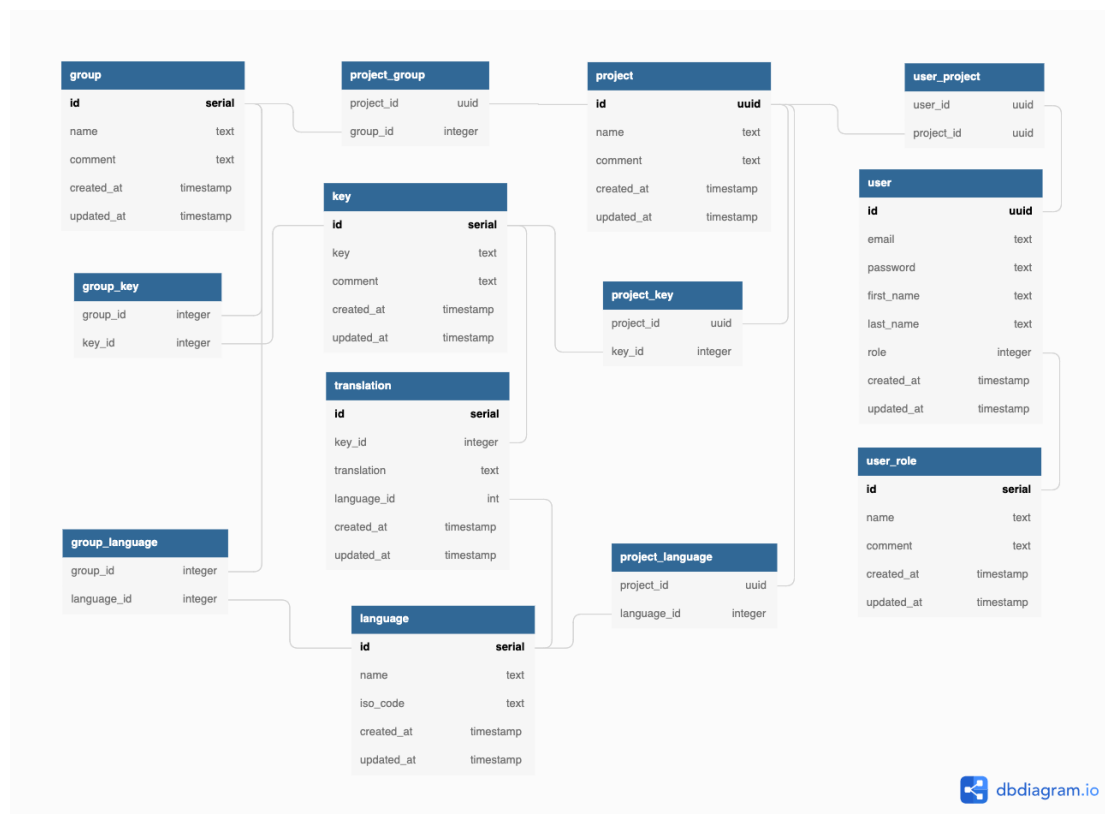
Diagram 1: DB schema v1

user table has roles which are defined in user_role table. A project can have users so we have join table between project and user table.

In the first version Projects and Groups are similar. Projects can have keys and groups. Groups can also have keys.

Keys have translations and translations have languages.

After realising during the development phase there were some issues with the first version of db schema. Projects and groups did not have a list of supported languages. Which made it hard to add a translation to a project or a group.



During the reviewing of the front-end application we came to realise that we don't really need to store keys into projects instead we can create some groups as templates and add the normal groups to projects, because group have the keys. Also from the integration point of view it makes sense to have keys in groups as the groups are converted into JSON modules on export. Now Projects have Groups, Groups have keys which have translations. Languages are still stored to projects because a language is global to a project and it stays same for all groups owned by the project.

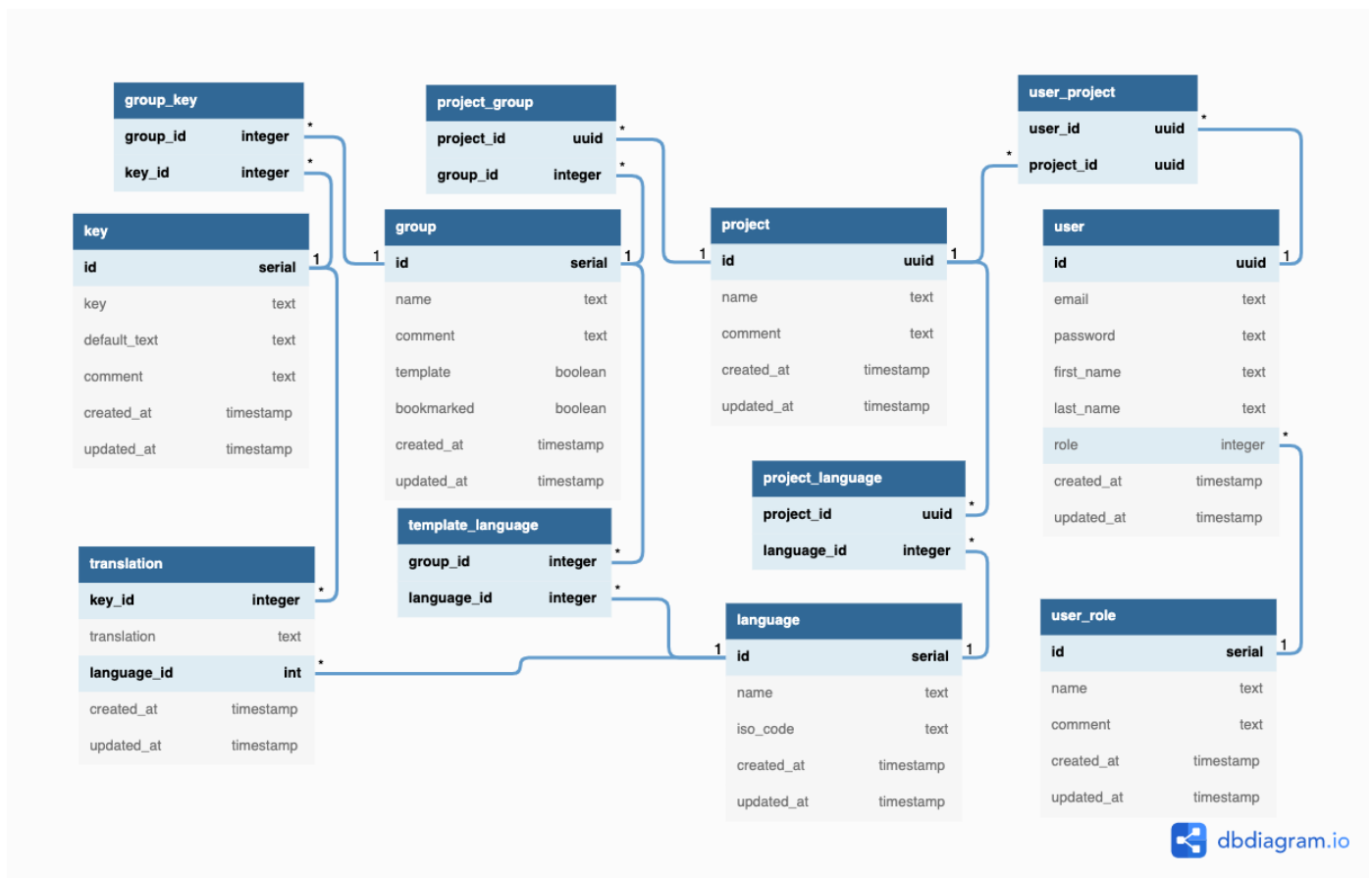


Diagram 3: DB schema final version

There is a new table `template_language`. This was needed when a boolean was added to groups stating if its a template. A template can have its own languages to store keys and translation. We have now templates so that, when a new project is created a template group can be copied with all its keys and translations to a new group in the project, which makes project initialisation more efficient.